MULT/IO I/O Controller
Technical Manual
Revision 4

April 1982

MORROW DESIGNS

MULT/IO Technical Manual

Revision 4

March 1982

Table of Contents

Table of Contents, Cont.

List of Figures

List of Tables

1. INTRODUCTION

The MULT/IO is a general purpose S-100 utility card that combines all the board level features needed to form the heart of a powerful interrupt driven, real time, multi-user system. Included on the board are:

Three 8250 programmable ACE serial devices (Asynchronous Communications Equipment) for communicating with RS-232 terminals or printers;

An 8259-A programmable interrupt controller (PIC) capable of resolving 8 levels of maskable, prioritized interrupts and of issuing 8080/8085/Z-80 CALL instructions as response for each level;

A CMOS real time clock/calender able to cause interrupts at software selectable intervals and with provision for battery back-up;

Three parallel ports (one input and two output) configured to plug directly into the ribbon cable connector of a parallel Diablo type 'Daisy Wheel' printer;

2 Kbyte of 2716 EPROM and 2 Kbyte of high speed static RAM-- both RAM and EPROM being bank selectable AND able to respond to all 24 S-100 address lines as defined in IEEE spec 696;

A power-on-jump option which allows 8 bytes of code to be executed from on-board EPROM during system power-on or reset.

The serial, parallel, clock and PIC devices on the MULT/IO are all I/O mapped-- that is, they are accessed through switch selectable I/O port addresses. These devices may be programmed to request service from the PIC based on a rich selection of status conditions. The 8259-A PIC can in turn issue to the CPU up to eight maskable, prioritized interrupt service routine call vectors. As the sole system I/O card, one MULT/IO board can be configured to support three terminals and a 'Daisy Wheel' printer while furnishing a real time, interrupt driven environment with all interrupt service routines optionally residing in on-board bank select RAM and EPROM. Alternatively, up to four MULT/IO cards may be combined to accommodate as many as twelve terminals with full interrupt support.

The on-board 8259-A interrupt controller may be jumpered to monitor any three vectored interrupt lines (S-100 bus lines 4-11) and can assert either the generalized interrupt request line (S-100 bus line 73) or any vectored interrupt line. Thus interrupts generated from off-board devices may be routed to the MULT/IO PIC using the vectored interrupt lines (Master Mode), or the MULT/IO PIC can send its interrupt requests over the vectored interrupt lines to some other interrupt controller (Slave Mode).

2. MULT/IO ARCHITECTURE

All devices on the MULT/IO, including RAM and EPROM, are associated with some S-1ØØ I/O port. In all, almost 3Ø distinct I/O registers are used to control the many device functions available on the board. Yet the MULT/IO takes up only 8 I/O port addresses. To understand how so many registers can be accessed through so few ports, it is useful to think of the port addressing scheme of the MULT/IO as 'bank-select I/O'. This is analogous to conventional bank-select memory schemes. Specifically, banks of registers are allowed to share the same block of consecutive I/O addresses while a dedicated I/O port is used to enable one bank and at the same time to disable all other similarly accessed banks.

The MULT/IO is divided into 4 I/O banks, called **groups**, with each group occupying the same 7 I/O port addresses. Three of these groups are used for the serial ports. The other group addresses the parallel ports, the clock, memory bank select and the interrupt controller (PIC).

Each group is accessed through ports BASE to BASE+6. Port address BASE+7 is the GROUP SELECT port, and is used to establish which of the four I/O groups will be active at any given time. By outputting the correct bit pattern to the GROUP SELECT port, the user enables the corresponding group for all subsequent I/O operations directed to ports between BASE and BASE+6. To enable a different group the user must output a different bit pattern to GROUP SELECT port BASE+7. While this port selection technique is extremely efficient in conserving I/O space, it does impose on the user the responsibility of keeping track of which I/O group is currently active.

2.1. GROUP SELECT PORT BASE+7

The Group Select port is a write-only register. Its functions do not vary with the selection of different groups. Besides being the Group Select port, it also controls bank select of on-board memory, enables the interrupt controller and parallel ports, and the printer restore bit. Thus, whenever a differant group is selected, care must be taken not to change the bits that control the other functions of the Group Select port.

Since this port is write-only, the last value output to this port must be kept in a location in memory that is known to all the software that needs to change the group select. Also, any interrupt routine that changes the currently selected group must restore it before exiting the service routine.

2.1.1. FUNCTIONS OF THE GROUP SELECT PORT

The following table outlines the bit assignment for the Group Select Port:

Table 2-1: GROUP SELECT PORT BASE+7

Data Bit	Function
Ø	This bit,and the next bit,control which
1	group is selected.
2	Memory Bank Select
3	Enable interrupt controller
4	Control printer restore (pin 13 of P4)
5	Enable parallel port output lines
6	Not used
7	Not used

The uses of bits 2-5 are described later in this manual.

The actual group assignments are determined by bits Ø and 1. Each group is selected by reading the current Group Select Data from memory, modifying bits Ø and 1, outputting the byte to the Group Select port and saving the data in memory. In the program examples used in this manual, the memory location for saving the current Group Select Data is called SELDAT.

The actual groups selected by bits Ø and 1 are defined in the table that follows:

Table 2-2: GROUP SELECT BITS

Bit Ø	Bit 1	Group	Group Description
Ø	Ø	Ø	Parallel Ports,199Ø Clock,8259-A PIC
Ø	1	1	Serial port 1
1	Ø	2	Serial port 2
1	1	3	Serial port 3

As an example of using the GROUP SELECT port, suppose that we want the I/O space taken up by the MULT/IO to extend from 8ØH to 87H, and that we wish first to read ACE serial device #2 and subsequently to read "DAISY PORT" Ø. In order to read the data received buffer of the second ACE serial device (serial device number 2), the user must first output SELDAT with a 1 in bit 1 and a zero in bit Ø to GROUP SELECT port 87H (to insure that I/O GROUP 2 is selected), and then input the desired data from port 8ØH (assuming the serial device has been properly initialized).

To read the parallel 'Daisy' port, we would first switch to I/O GROUP Ø by outputting SELDAT with zeros in both bits Ø and 1 to port 87H, and then input the desired data from port 8ØH. The important thing to note here is that the function of I/O port 8ØH in our example changes from a serial device data register to a parallel device status register depending on the last byte that we output to the GROUP SELECT port. It is important not to change data bits 2-5 when outputting group select data to port BASE+7.

2.1.2. GROUP PORT ASSIGNMENTS

Thispage contains a general map of the port assignments within the groups. Specific details, such as control bit assignments, are described in detail in the sections that describe each device.

Table 2-3: GROUP Ø

	INPUT	OUTPUT
BASE	DAISYØ IN	DAISYØ OUT
BASE+1	not used	DAISY1 OUT
BASE+2	CLOCK IN	CLOCK OUT
BASE+3	not used	not used
BASE+4	8259-A AØ=Ø REGISTER	8259-A AØ=Ø REGISTER
BASE+5	8259-A AØ=1 REGISTER	8259-A AØ=1 REGISTER
BASE+6	not used	not used

Table 2-4: GROUPS 1, 2, & 3
(825Ø ACE Serial I/O Ports)

	INPUT	OUTPUT
BASE	RECEIVE BUFFER/LSB BAUD	TRANSMIT BUFFER/LSB BAUD
BASE+1	INTERRUPT ENABLE/MSB BAUD	INTERRUPT ENABLE/MSB BAUD
BASE+2	INTERRUPT IDENTIFY	not used
BASE+3	LINE CONTROL REGISTER	LINE CONTROL REGISTER
BASE+4	MODEM CONTROL REGISTER	MODEM CONTROL REGISTER
BASE+5	LINE STATUS REGISTER	not used
BASE+6	MODEM STATUS REGISTER	not used

NOTE: AN OUTPUT TO BASE+7 WILL ALWAYS **ASSIGN** AN I/O GROUP BUT HAS NO FUNCTION WITHIN ANY GIVEN I/O GROUP.

2.2. SELECTING I/O PORT ADDRESS

The base address of the MULT/IO ports is selected using Switch 7B. This switch is set to match the upper 5 bits of the port address (A3-A7). The BASE port can be located at any 8 byte boundary, starting at port Ø and ending at port F8H. The relationship between switch number and address bit is illustrated below:

SWITCH 7B

number	address bit
2	A7
3	A6
4	A5
5	A4
6	A3

Setting a switch ON matches a zero, and OFF matches a 1. For example, with all switches OFF, the MULT/IO will occupy I/O addresses F8H to FFH; with all switches ON it would occupy ports Ø through 7.

2.3. RAM AND EPROM-- GENERAL

The MULT/IO is equipped to handle four kilobytes of high speed static RAM or four kilobytes of 2716 EPROM or a combination of each. This memory occupies two sockets at 5D and 6D on the board. The left hand socket, 5D, is called RØ, and is assigned the first 2K of address space, and the one to the right of it, 6D, is called Rl and is assigned the last 2K of the four kilobyte region.

This memory always functions as bank select memory (see Bank Selection), and is addressed as a 4K unit.

No wait state is generated when accessing MULT/IO memory, which is capable of running solid at up to 6 megahertz. There is no provision for generating wait states as a user option. If special uses require wait states, a Programmable Logic Array would have to be special ordered from Morrow Designs.

2.3.1. ADDRESSING RAM AND EPROM

The MULT/IO memory may be addressed to any 4K boundary in the 64K address region, or in the 16 megabyte address region of the full IEEE 696 specifications. To select an address, in either region, the higher four bits of the 16 bit address are selected by setting the switches of 3-6 of 1ØB. The additional 8 bits of extended addressing are covered in the next section.

Table 2-5: MEMORY ADDRESSING
Switch Bank 1ØB

Address Bit	Switch #
A15	3
A14	4
A13	5
A12	6

ON = Ø and OFF = 1

EXAMPLE: To set RAM to begin at CØØØH, switches 3 and 4 should be placed in the "OFF" position, and switches 5 and 6 should be placed in the "ON" position. This will cause RAM to occupy address space from CØØØH to CFFFH. The memory at RØ will range from CØØØH to C7FFH, and the memory at Rl will begin at C8ØØH and end at CFFFH.

The following table gives all of the 16 possible settings of the RAM/EPROM address switch at 1ØB and the corresponding beginning and ending addresses of on-board RAM and EPROM.

Table 2-6: ADDRESS SETTINGS
(in first 64K block)

A15 1ØB-3	A14 1ØB-4	A13 1ØB-5	A12 1ØB-6	RØ BEGIN	RØ END	Rl BEGIN	Rl END
ON	ON	ON	ON	ØØØØ	Ø7FF	Ø8ØØ	ØFFF
ON	ON	ON	OFF	1ØØØ	17FF	18ØØ	1FFF
ON	ON	OFF	ON	2ØØØ	27FF	28ØØ	2FFF
ON	ON	OFF	OFF	3ØØØ	37FF	38ØØ	3FFF
ON	OFF	ON	ON	4ØØØ	47FF	48ØØ	4FFF
ON	OFF	ON	OFF	5ØØØ	57FF	58ØØ	5FFF
ON	OFF	OFF	ON	6ØØØ	67FF	68ØØ	6FFF
ON	OFF	OFF	OFF	7ØØØ	77FF	78ØØ	7FFF
OFF	ON	ON	ON	8ØØØ	87FF	88ØØ	8FFF
OFF	ON	ON	OFF	9ØØØ	97FF	98ØØ	9FFF
OFF	ON	OFF	ON	AØØØ	A7FF	A8ØØ	AFFF
OFF	ON	OFF	OFF	BØØØ	B7FF	B8ØØ	BFFF
OFF	OFF	ON	ON	CØØØ	C7FF	C8ØØ	CFFF
OFF	OFF	ON	OFF	DØØØ	D7FF	D8ØØ	DFFF
OFF	OFF	OFF	ON	EØØØ	E7FF	E8ØØ	EFFF
OFF	OFF	OFF	OFF	FØØØ	F7FF	F8ØØ	FFFF

If only the lower 16 address lines are used (for a 64K address space), the extended addressing feature must be disabled. This is done by setting switch 1 of 1ØB to the ON position and removing the IC (25LS2521) at location 3D (next to the extended address switch at 2D) from its socket.

2.3.2. EXTENDED ADDRESSING

Extended addressing as applied to S-1ØØ memory devices is simply the ability of memory to decode more than 16 address bits in order to become selected. The 4K block of RAM/EPROM on the MULT/IO may be switched to decode 24 rather than 16 address lines-- the extra 8 address lines are defined by IEEE specification 696. This extended addressing feature allows the RAM/EPROM on the MULT/IO to occupy any even 4K block within a 16 Megabyte address space.

To enable this extra decoding circuitry, switch 1 of DIP switch 1ØB must be placed in the OFF position. Since many CPU boards currently in use do not generate address lines A16 - A23, many users will wish to disable the extended addressing circuitry of the MULT/IO. This is done simply by setting switch 1 of DIP switch 1ØB to the ON position. It is recommended that when running the board in non-extended mode the IC at location 3D (25LS2521) be removed from its socket.

With extended addressing enabled (switch 1 of 1B OFF), the DIP switch at location 2D determines the 64K segment wherein the 4K of on-board RAM/EPROM will reside. The following table illustrates the switch settings of DIP switch 2D and their corresponding extended address bits. The S-1ØØ bus pin numbers assigned by the IEEE specification 696 to these extended address bits are given in parentheses.

Table 2-7: EXTENDED ADDRESSING
DIP Switches 2D and 1ØB

Extended Address Bit	S-1ØØ Bus Pin #	DIP Switch 2D Switch #
A23	(16)	1
A22	(17)	2
A21	(15)	3
A2Ø	(59)	4
A19	(61)	5
A18	(62)	6
A17	(63)	7
A16	(64)	8

DIP Switch 1ØB-1
must be OFF to enable
extended addressing

and ON to disable
with chip 3D removed

ON = Ø
OFF = 1

Example: To set RAM/EPROM to begin at 8ØCØØØH, set switch 1 of 1ØB OFF to enable extended addressing, set the lower 16 bits (the CØØØ part of this address) on DIP switch 1ØB as per the instructions on the previous page, and set switch 1 of DIP switch 2D OFF, and switches 2 - 8 ON. Set in this way, on-board EPROM/RAM will respond to all memory accesses from 8ØCØØØH to 8ØCFFFH. When so addressed, RAM/EPROM will NOT respond to memory accesses to the area from ØØCØØØH to ØØCFFFH, and so would in effect be permanently disabled in any system incapable of generating extended addresses.

2.3.3. BANK SELECTION

The RAM/EPROM block on the MULT/IO is bank select memory--that is, an I/O instruction can cause the memory block to become enabled or disabled. Bit 2 of port BASE+7, the Group Select Port, controls the bank select. The effect of outputting a zero or one in this bit position is to turn on or off the RAM/EPROM. The choice of which value to use (one or zero) is dependent on the way the board is set to respond after RESET/ or POJ/.

Switch 1ØB-2 allows the user to determine whether MULT/IO RAM/EPROM will be selected or not after system power-up or reset. The setting of this switch also determines whether data bit 2 will be active high or active low when an output instruction is directed to port BASE+7. If Switch 1ØB-2 is in the ON position, then the MULT/IO RAM/EPROM bank will be enabled upon system power-up or reset, and data bit 2 will have to be low or 'Ø' for Group Select port BASE+7 to enable memory, and high or '1' to disable. If Switch 1ØB-2 is OFF, the MULT/IO RAM/EPROM bank will be disabled upon system power-up or reset, and will not be accessible until an output is made to port BASE+7 with data bit 2 a '1' or high. The following table reiterates this:

Table 2-8: BANK SELECT AND SWITCH 1ØB-2

Position of Switch 1ØB-2	Condition of RAM/EPROM after RESET/ or POJ/	Bank Select	Bank Deselect
ON	enabled	Ø	1
OFF ✓	disabled	1	Ø

The bank select value is output along with the Select Data to port BASE+7 to enable MULT/IO memory. The bank deselect value disables memory.

When disabled by bank de-selection, MULT/IO RAM/EPROM will 'disappear' from the bus, and so will not interfere with other system memory occupying an identical address. Therefore other bank select memory boards may be swapped in and out of memory along with MULT/IO RAM/EPROM. Of course, memory cards which are to be swapped in and out along with MULT/IO RAM/EPROM must themselves be capable of being disabled through some software mechanism.

EXAMPLE: To show how the MULT/IO memory would be enabled after a RESET/, when it was disabled because Switch 1ØB-2 was OFF, a 1 in bit 2 (1ØØB or 4H) is output to the Group Select Port, and the new value of Select Data is saved.

```
bank:       lda  seldat        ;recall old group select data
            ori  4             ;set bank select bit high
            sta  seldat        ;save the modified select data
            out  base+7        ;send to group select port
            ret                ;only bank select has changed
```

CAUTION!

The Group select Port, BASE+7, is a write-only port with multiple functions. Whenever any bit is changed, the appropriate bit in SELDAT should be set or cleared and saved. The example above shows how this may be done.

2.3.4. PHANTOM

SWITCH 1 of 1ØB is the Phantom enable switch. When placed in the OFF position, the MULT/IO will ignore bus pin 67, or "Phantom". In the ON position, this switch causes the RAM/EPROM section of the MULT/IO board to become disabled and logically removed from the system bus whenever bus pin 67 is at a low logic state. When pin 67 becomes high, MULT/IO memory will be enabled if it was previously bank selected.

Certain systems rely on the Phantom line to temporarily disable RAM memory in order to execute from ROM a special system start-up routine. Once this routine is executed, the ROM holding the routine vanishes and the Phantom line returns high to allow RAM memory to be accessed. MULT/IO memory is compatible with such a scheme.

The PHANTOM/ line is also used during interrupt acknowledge sequences. While the 8259-A PIC is placing the low and high bytes of the vector address on the bus, PHANTOM/ can be made true to disable memory. This is because during the first cycle of interrupt acknowledge Z-8Ø's assert INTA/ which disables memory boards. However, during the next two cycles, the Z-8Ø will not assert INTA/. The MULT/IO board can be configured to assert PHANTOM/ during these two cycles. Please refer to the section on configuring the MULT/IO for the PIC.

2.3.5. POWER ON JUMP

Switch 1ØB-7 controls the power-on jump circuitry of the MULT/IO. When placed in the ON position, this switch will cause the MULT/IO to force the host processor to execute the last 8 instructions of a MULT/IO EPROM.

To use the Power on Jump feature, there must be at least one EPROM in either RØ or R1, the two MULT/IO memory sockets. Switch 1ØB-2 must be ON, so that the memory is enabled on RESET/ (see Bank Select above). Then, Switch 1ØB-8 must be set tochoose which of the two memories, RØ or R1, will be read. Setting 1ØB-8 ON selects RØ, and turning it OFF selects R1 (the memory chip at 6D, on the right).

When the MULT/IO power on jump is used, the last 8 bytes of an EPROM will be read. Typically, the last three bytes will be a jump instruction to the user's bootstrap routine.

NOTE: In order to use the power on jump, all four of these conditions must be met:

There must be an EPROM on the MULT/IO with instructions in the last eight bytes;

This EPROM must be selected by using switch 1ØB-8;

The MULT/IO memory must be enabled on RESET/, that is Switch 1ØB-2 must be ON;

The power on jump switch 1ØB-7 must be ON.

3. SERIAL PORTS

The MULT/IO has three 8250 programmable Asynchronous Communications Elements (ACE's) which can be connected to RS-232 devices via three 26 pin ribbon cable connectors. Each ACE has an I/O group dedicated to it-- namely, GROUPS 1, 2 and 3. The ACE's are completely programmable and must be initialized in software before they can be used. Initialization includes setting the baud rate, word length, parity, number of stop bits, and interrupt conditions.

All three ACE's are configured as Data Communications Equipment (DCE) from the factory, and so may be connected with standard RS-232 CRT terminals and printers. All may be re-strapped to be used as Data Terminal Equipment (DTE) if they need to be connected to modems or other computers.

Each ACE can be programmed to generate an interrupt in response to up to ten conditions (e.g., data available, transmitter buffer empty, etc.). The interrupt is sent directly to the MULT/IO PIC which can in turn pass it on to the host CPU. The interrupt handling routine can then interrogate the interrupt status register of the ACE responsible for generating the interrupt, and is thus able to determine the precise cause of the interrupt.

The following chart describes the ACE devices on the MULT/IO, including the location of the 8250 on the circuit board, the location of the 26 pin ribbon cable connector associated with each ACE, the I/O GROUP controlling each ACE, and the interrupt level assigned to each device by the 8259-A PIC.

Table 3-1: ACE I/O GROUP DESCRIPTION

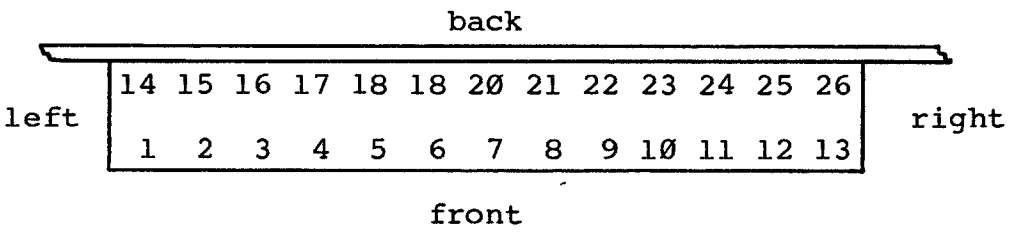
	I/O GROUP #	26-pin connector	Board location	Interrupt Level
ACE # 1	1	P1	2C	3
ACE # 2	2	P2	2B	4
ACE # 3	3	P3	2A	5

P1 is the connector on the top left corner of the board; P2 and P3 are the next two connectors to the right of P1.

3.1. CONFIGURING SERIAL CONNECTORS

The pins on ribbon cable connectors P1-P3 are numbered so that the first 25 pins correspond exactly to the numbering of a standard DB-25 connector (i.e., first row left to right, 1 to 13, second row left to right, 14 to 25). This makes it a simple matter to attach each ACE to a serial device-- cables with flat ribbon cable connectors at one end and DB-25 connectors on the other are available off the shelf from many vendors.

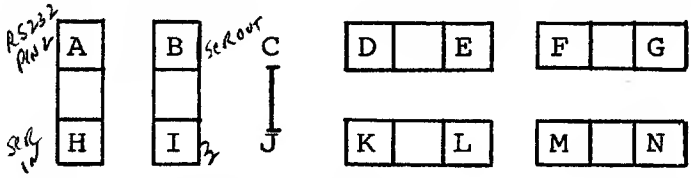
Figure 3-1: P1-P3 Connector Pinout



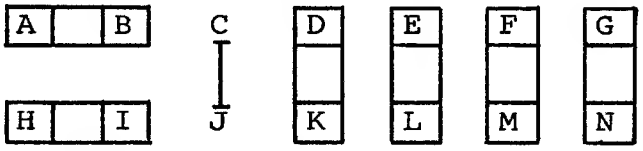
Directly below each 26 pin connector is an array of 7 pairs of jumper headers labeled J1, J2, and J3. They are used to configure P1 through P3 as modem (factory strapped) or as terminal. Six slip-on connectors are used to supply the standard arrangements of pin assignments. Other non-standard assignments may be made using wire-wrap. The figures that follow show the two normal configurations of J1, J2 and J3.

Figure 3-2: SERIAL CONFIGURATION JUMPERS

Serial Port as modem (Data Communication Equipment), standard



Serial Port as terminal (Data Terminal Equipment)



The two serial configurations represent the opposite ends of a connecting cable: transmit data from one end goes to receive data at the other end, and request to send is connected with clear to send, etc. Normally, computers are configured as modems (for connection with terminals). In order to tie two computers together, you would configure a serial port on one computer as a terminal. This correctly transposes all the handshaking and transmit/receive lines.

All of the active lines on the 26 pin connectors, with the exception of pins 1 and 7 which are tied to ground, are brought to the jumpers. In the same way, the transmit/receive and control pins of the 825Ø ACE are brought to the jumpers. This allows the easy interchange of signals when configuring each 825Ø as terminal or modem. The illustrations on the previous page show the standard configurations of these jumpers. The following table describes the connections of pins to the jumper.

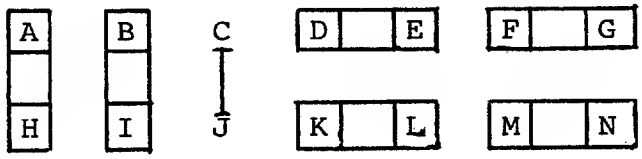
Table 3-2: ACE JUMPER CONNECTIONS

26 pin connector P1, P2 or P3	J1, J2 or J3 jumper pins	pin of 825Ø ACE			signal or DCE name
		2C	2B	2A	
2	A				RXD
	B	11	11	11	sout
8	C	*	*	*	DCD
	D	33	33	33	dtr
6	E				DTR
	F	32	32	**	rts
5	G				RTS
	H	1Ø	1Ø	1Ø	sin
3	I				TXD
*	J	38	38	38	rlsd
2Ø	K				DSR
	L	37	37	37	dsr
4	M				CTS
	N	36	36	36	cts

* These pins are hardwired together.

** RS-232 line 4 (request to send) is implemented only on ACE #1 and 2, NOT on ACE # 3. Also, Ring Indicator, RS-232 pin 22, is not implemented. Though this function has a dedicated line on the 825Ø ACE and has its own status bit in the Modem Status Register, the 825Ø RI pin (31) is tied high on the MULT/IO, and so sampling it would be meaningless.

Here is an illustration of J1 configured as a modem (as it comes from the factory):



3.2. PROGRAMMING THE 8250

Any 8250 ACE device on the MULT/IO can be accessed only if its I/O GROUP is currently selected. Once a 1, 2 or 3 has been output to GROUP SELECT port BASE+7, ACE device number 1, 2 or 3 can then be accessed. Each ACE contains internal 8 bit registers which occupy the first 7 I/O ports of the MULT/IO I/O space--that is, ports BASE to BASE+6. The list below identifies all the internal registers of the 8250 and the I/O port address assigned to those registers by the MULT/IO.

It should be noted that the first two ports, BASE and BASE+1 have dual use. When the ACE is initialized, it is necessary to specify the baud rates. This is done by first setting up the LINE CONTROL REGISTER (BASE+3) with bit 7 set to 1. This makes the first two 8250 registers the low and high byte of the baud rate divider. After outputting the divider to these two registers, the line control word is again output to BASE+3 with bit 7 reset (to 0). This switches the first two registers to their normal use. Baud rates are described in the following section.

Table 3-3: REGISTERS OF THE 8250 ACE

I/O PORT	OPERATION	bit 7 of BASE+3	8250 ACE Register
BASE	Write	0	Transmitter Buffer
BASE	Read	0	Receive buffer
BASE	Read/Write	1	Baud rate divisor - low byte
BASE+1	Read/Write	0	Interrupt enable mask
BASE+1	Read/Write	1	Baud rate divisor - high byte
BASE+2	Read	x	Interrupt ID register
BASE+3	Read/Write	x	Line Control Register
BASE+4	Read/Write	x	MODEM Control Register
BASE+5	Read/Write	x	Line Status Register
BASE+6	Read/Write	x	MODEM Status Register

For a complete description of these registers, refer to the data manual on the 8250. x means "don't care".

> NOTE: Auxiliary OUT1 and OUT2 are not available in MODEM control register; also, bits 2 and 6 of MODEM status register, Ring Indicator, are meaningless.

3.2.1. BAUD RATE

The 8250's on the MULT/IO have been hard wired so that the baud rate for data coming in is the same as for data going out. The crystal used to provide the reference frequency for the three ACE devices on the MULT/IO is 1.8432 MHz. The data sheets give a broad sample of the divisors which must go into the Divisor Latch in order to generate the most common baud rates, and generally any baud rate may be generated from DC (a zero in the divisor

latch-- this will inhibit all data transmission) up to 56,ØØØ baud. The formula for determining the divisor constant to produce a given baud rate is :

DIVISOR = 1.8432 M/(BAUD RATE X 16)

Although in most applications the user will simply look up the baud rate divisor in the data sheet table, there are instances when 'odd ball' baud rates may be useful-- if, for example, an ACE is being used solely to generate interrupts at timed intervals based on the Transmitter Holding Register Empty interrupt (see Serial Device Interrupts).

The following is a list of divisor latch constants for standard baud rates. The baud rate is given in decimal, followed by the divisor in decimal. The next two values are the hex numbers actually output to BASE and BASE+1, when bit 7 of BASE+3 is a 1.

Table 3-4: DIVISOR LATCH CONSTANTS FOR STANDARD BAUD RATES

Baud Rate (Decimal)	Divisor (Decimal)	Low Byte (Hex)	High Byte (Hex)
75	1536	Ø	6
11Ø	1Ø47	17	4
15Ø	768	Ø	3
3ØØ	384	8Ø	1
6ØØ	192	CØ	Ø
12ØØ	96	6Ø	Ø
24ØØ	48	3Ø	Ø
48ØØ	24	18	Ø
96ØØ	12	C	Ø
192ØØ	6	6	Ø
384ØØ	3	3	Ø
56ØØØ	2	2	Ø

3.3. INITIALIZATION

Though the reset pin (MR) of each 825Ø will be asserted during power-on or reset, no assumptions should be made about the contents of any 825Ø register unless that register has been initialized. Keep in mind that an on-board ACE cannot be accessed, far less initialized, unless its I/O group is selected. Furthermore, the Line Control, Modem Control, Interrupt Enable and Divisor Registers will normally have to be initialized before any data can be transferred to or from an 825Ø.

The following three software routines are brief samples of how a MULT/IO ACE device could be driven in a CP/M* type environment. All these routines adhere to CP/M* I/O protocol. The INIT

* CP/M is a trademark of Digital Research.

routine sets up ACE # 1 to run at 9600 baud with an 8 bit word, no parity and 2 stop bits. The Interrupt Enable Register will be set to generate no interrupts, and the Modem Control Register will be ignored. This initialization would be appropriate for most RS-232 CRT terminals in a non-interrupt driven environment. Assume that the MULT/IO I/O has been set to begin at 48H. The cluster of assembler directives (equ's) at the beginning of these routines establish constants which hold for all 3 specimen routines. The comments included with these routines may be used as a general flow analysis of ACE programming.

3.3.1. SAMPLE SERIAL I/O ROUTINES

```
groupl  equ     1       ;code for first ACE (attached to J1)
base    equ     48h     ;base I/O address set by SW-8C
grpctl  equ     base+7  ;board group control port
dll     equ     base    ;ACE baud rate divisor (lsb)
dlm     equ     base+1  ;ACE baud rate divisor (msb)
ier     equ     base+1  ;ACE interrupt enable register
lcr     equ     base+3  ;ACE line control register
lsr     equ     base+5  ;ACE line status register
rbr     equ     base    ;ACE receiver buffer register
thr     equ     base    ;ACE transmitter holding register
dlab    equ     80h     ;divisor latch access bit
thre    equ     20h     ;line status register THRE bit
dr      equ     1       ;line status register DR bit
baudl   equ     12      ;divisor latch low byte-- 9600 baud
baudh   equ     0       ;divisor latch high byte-- 9600 baud
wls0    equ     1       ;word length select bit 0-- 8 bit word
wls1    equ     2       ;word length select bit 1-- 8 bit word
stb     equ     4       ;stop bit count-- 2 stop bits
imask   equ     0       ;interrupt mask-- disable all
;
```

SAMPLE SERIAL I/O ROUTINES

```
;The following routine initializes the ACE as described above
;
init:   mvi     a,groupl ;set up desired I/O group
        out     grpctl   ;select first serial device
                         ;next set up format and set dlab
        mvi     a,dlab+wlsØ+wlsl+stb
        out     lcr      ;base reg is now lsb baud rate reg
        mvi     a,baudl  ;low byte of baud rate constant
        out     dll      ;into low baud rate register
        mvi     a,baudh  ;high byte of baud rate constant
        out     dlm      ;into high baud rate register
                         ;set up format and clear dlab
        mvi     a,wlØ+wll+stb
        out     lcr      ;into line control register
        xra     a        ;zero register a
        out     lsr      ;clear data available flag in line status
        mvi     a,imask  ;interrupt mask set up
        out     ier      ;base+1 now interruptmask- not baud
        ret              ;end of initialization routine
;
;The following routine will return in the accumulator any new
;character typed to ACE # 1
;
conin:  mvi     a,groupl
        out     grpctl   ;put a 1 into MULT/IO GROUP SELECT port
                         ;make sure dlab is cleared
        mvi     a,wlsØ+wlsl+stb
        out     lcr      ;make base port the ACE data register
coninl: in      lsr      ;get line status register
        ani     dr       ;any new data from terminal?
        jz      coninl   ;if no then keep waiting
        in      rbr      ;get data
        ani     7fh      ;strip off bit 7 of input character
        ret              ;return with data in accumulator
;

;The following routine will output the character in Register C
;to ACE # 1
;
conout:  mvi    a,groupl
         out    grpctl   ;put a 1 into MULT/IO GROUP SELECT port
                         ;make sure dlab is low
         mvi    a,wlsØ+wlsl+stb
         out    lcr      ;make base port the ACE data register
conoutl: in     lsr      ;get line status
         ani    thre     ;is ACE ready to transmit?
         jz     conoutl  ;if not then keep waiting
         mov    a,c      ;transfer data from reg c to reg a
         out    thr      ;output character typed from terminal
         ret             ;return to calling program
;
```

```
;The following routine will return an FF in the Register A if ACE
;device # 1 has received a new character (i.e., DR is set in the
;ACE line status register).  Otherwise, return a Ø.
;
status:  mvi     a,groupl
         out     grpctl   ;put a 1 into MULT/IO GROUP SELECT port
         in      lsr      ;get line status
         ani     dr       ;check DR bit
         rz               ;return if reg a is zero-- no character
         mvi     a,Øffh   ;ff into reg a since character is ready
         ret
```

In the above examples, it should be noted that the GROUP SELECT port is re-initialized at the beginning of every routine. This is done to insure against inadvertently sending serial I/O instructions to the clock, parallel ports or interrupt controller of the MULT/IO.

In this example please note that before accessing the ACE data register, the format word is sent again to the Line Control Register. This is done so that port BASE of GROUP 1 will be interpreted as a data port rather than as a divisor port. This guards against a situation such as losing access to the console device due to writing of the Divisor Latch (from a monitor or front panel, for example) without subsequently clearing bit 7 of BASE+3, DLAB. This precaution may be unnecessary in most non-developement systems.

3.4. SERIAL DEVICE INTERRUPTS

The three 825Ø ACE devices on the MULT/IO each have a dedicated interrupt request line on the 8259 PIC. The chart below desribes the PIC interrupt level assigned to each ACE:

Table 3-5: ACE INTERRUPT ASSIGNMENTS ON 8259 PIC

Serial Device	PIC Interrupt Request Line
ACE # 1 (I/O Group 1)	IR3
ACE # 2 (I/O Group 2)	IR4
ACE # 3 (I/O Group 3)	IR5

3.4.1. ACE INTERRUPT PROGRAMMING

As explained in the data sheet on the 825Ø, each ACE device can be programmed to generate an interrupt on any of four general conditions. These conditions are, in order of descending priority: Receiver Line Status, Received Data Available, Transmitter Holding Register Empty, and Modem Status. The Received Data Available and the Transmitter Holding Register Empty interrupts can be identified directly from the Interrupt ID Register of the source ACE.

The remaining two interrupts must use the Interrupt ID Register to point to either the Receiver Line Status Register or the Modem Status Register. These two registers each have four interrupt flags which can be read to identify the source of an ACE generated interrupt. (The third interrupt of the Modem Status Register-- The Trailing Edge of Ring Indicator, or TERI-- is not usefully supported by the MULT/IO, since the Ring Indicator line of each ACE is tied to +5V.)

Because the 825Ø prioritizes its interrupts, the Interrupt ID Register will 'freeze' the highest priority interrupt pending by ignoring all further interrupts until the previous interrupt has been serviced. For detailed information of the interrupt structure of the 825Ø see the data sheets.

When using the 825Ø's ACE devices on the MULT/IO to generate interrupts, it is advisable to set the 8259-A PIC to operate in level mode, rather than edge mode. In edge mode, it is possible under certain circumstances for an ACE generated interrupt to be 'lost'-- that is, to go unrecognized. The 825Ø produces one low going edge for each interrupt produced. If the 8259-A PIC is currently servicing a different 825Ø interrupt, it will miss the edge and be unable to detect that the line is now low. Using level mode avoids this.

4. PARALLEL DAISY-WHEEL PRINTER PORT

The MULT/IO contains parallel I/O ports configured to accomodate a standard DIABLO type daisy wheel R/O printer. These ports are brought out to the 5Ø pin ribbon cable connector at P4 for easy attachment to a Diablo style printer. The pin assignments of P4 correspond exactly to those of an internal Diablo 5Ø conductor flat cable connector, so simply tying the Diablo to the MULT/IO via a ribbon cable with female sockets at either end is the only hardware requirement for interfacing the two devices.

Altogether, two latched output ports (plus an extra latched output bit) and one transparent input port are used to communicate with the Daisy Wheel printer. Of course, these ports may be used with practically any parallel device (e.g., a Centronics style printer or a keyboard) provided that the I/O lines are properly routed from the MULT/IO connector at P4 to the target device. This additional cabling burden is standard in parallel I/O interfacing, and so should not be considered as a major disadvantage by those using the DAISY PORT with a non-Diablo parallel device.

4.1. PARALLEL PORT DESCRIPTION

The MULT/IO DAISY PORT occupies I/O ports BASE and BASE+1, both within I/O GROUP Ø. Bit 5 of the Group Select Port (BASE+7) enables the output ports. A single input line (BASE+Ø bit 5, or the Print Wheel Ready line when interfacing with a Daisy Wheel printer) is, after going to the DAISY PORT, inverted and then brought to IRQ 6 of the 8259-A interrupt controller, and so can be used to generate an interrupt whenever it goes to a low logic state.

BASE+7 bit 5 enables all DAISY PORT output lines. If this bit is low, all output lines controlled by I/O ports BASE and BASE+1 will remain in a high impedance state regardless of other software commands.

The eight input lines brought to DAISY PORT BASE are also pulled up to +5V through 18Ø Ohms (nominal), and so may be used with open-collector devices. These eight input lines are inverted by an input buffer, and so if left unconnected will appear to software to be low.

The parallel ports have no special facility for generating a strobe on output or latching a strobe on input. All data lines operate as levels, so strobes must be generated in software.

The following page depicts the parallel lines available on the MULT/IO, including the I/O port and bit number controlling each line and the function assigned to each line on a standard parallel Diablo type interface. Remember that these functions have no inherent meaning to the MULT/IO, which simply sees so many latches, and so do not preclude interfacing the MULT/IO with parallel devices other than Daisy Wheel printers.

Table 4-1: DAISY PORT SIGNALS AND I/O MAP

I/O GROUP Ø

I/O Port	Data Bit	MULT/IO and Diablo Pin #	Diablo Function
Input BASE	Ø	4	End of Ribbon (-)
	1	3	Paper Out (-)
(these 8	2	5	Cover Open (-)
input lines	3	34	Paper Feed Ready (-)
pulled up	4	26	Carriage Ready (-)
to +5V by	5	27 *	Print Wheel Ready (-)
@18Ø Ohms	6	12	Check (-)
& inverted)	7	28	Printer Ready (-)
Output BASE	Ø	46	Data Bit 9 (256) (-)
	1	1	Data Bit 1Ø (512) (-)
	2	9	Data Bit 11 (1Ø24) (-)
	3	1Ø	Data Bit 12 (2Ø48) (-)
	4	15	Paper Feed Strobe (-)
	5	17	Carriage Strobe (-)
	6	21	Print Wheel Strobe (-)
	7	23	Ribbon lift (-)
Output BASE+1	Ø	37	Data Bit 1 (1) (-)
	1	36	Data Bit 2 (2) (-)
	2	39	Data Bit 3 (4) (-)
	3	33	Data Bit 4 (8)(-)
	4	4Ø	Data Bit 5 (16) (-)
	5	42	Data Bit 6 (32) (-)
	6	43	Data Bit 7 (64) (-)
	7	45	Data Bit 8 (128) (-)
Output BASE+7	4	13	Restore (-)

*In addition to being associated with bit 5 of Input Port Base, pin number 27 of P4 (the Diablo Print Wheel Ready line) is also connected through an inverter to Interrupt Request line 6 (pin 24) of the 8259-A PIC. Thus this line may be used to generate an interrupt whenever any external device brings it low (e.g., when the print wheel is ready).

The following lines on MULT/IO connector P4 are tied to ground as prescribed by the Diablo Interface:

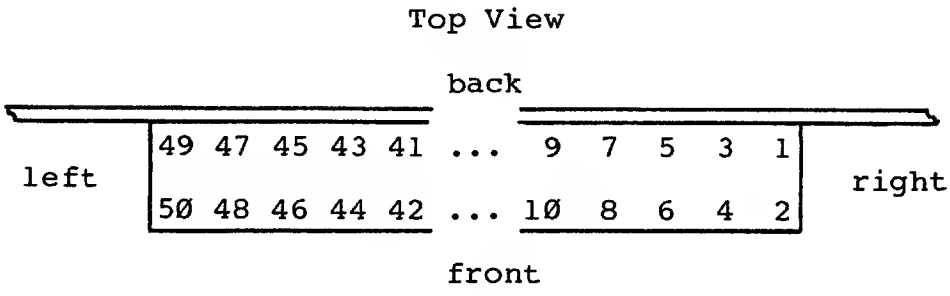
2, 8, 11, 14, 16, 18, 2Ø, 22, 25, 3Ø, 31, 32, 35, 38, 41, 44, 47.

Line 24, defined by Diablo as Select (-), is also grounded.

Line 48 of MULT/IO connector P4 is defined by Diablo as +5V (Reference Out). This line is not used by the MULT I/O.

Unimplemented (left floating) are lines 6, 7, 29, and 5Ø.

Figure 4-1: DAISY PORT P4 CONNECTOR PINOUTS



4.2. PROGRAMMING THE DAISY PORT

As with all I/O devices on the MULT/IO, the user must be careful, when accessing the DAISY PORT, to initialize the correct I/O group-- in this case, GROUP Ø. Once the proper I/O Group has been selected, all data output from the CPU to the parallel ports is latched. By latched is meant that the data output to a parallel port will appear on the appropriate pins on the P4 connector, and will remain there until either different data is output to the port in question or until Driver Enable (bit 5 of Select Group Port BASE+7) is brought low. When this occurs, all 17 parallel output pins of connector P4 will enter a high impedance state.

The 8 input lines of the DAISY PORT are available to the CPU through an inverter, so that when an input instruction is directed at DAISY PORT Ø, the CPU will read the complement of whatever data is on the appropriate lines of connector P4 at the time the input instruction is executed. There is no provision for strobing data into the parallel input buffer for later examination after the data to be read has gone away.

The MULT/IO DAISY PORT inverts its input lines but does NOT invert its output lines. Daisy Wheel printers use negative logic, so that a low signal is taken as active. Thus to assert, or make active, any output line when talking to a Daisy Wheel printer, the software must put the line low. Input lines from a Daisy Wheel printer, on the other hand, are inverted in hardware, and so will appear to software to be active high.

4.2.1. GENERATING AN OUTPUT STROBE

To generate an output strobe off any of the parallel output ports on the MULT/IO, it is necessary to use a software mask. This means that the line to be strobed must be output three times in succession, changing state each time, while the data lines associated with the same port must be allowed to remain unchanged. For example, to output a strobe going high-low-high on bit 7 of port BASE without changing the other 7 bits being output from that port, the following routine could be used:

```
lda   seldat     ;get old select data
ani   ØFCh       ;select group Ø, w/o modifying other bit
sta   seldat     ;save new select data
mvi   a,c        ;original data into register A
ori   8Øh        ;preserve data but bring bit 7 high
out   base       ;output data with bit 7 high
ani   Ø7fh       ;preserve data but bring bit 7 low
out   base       ;output data with bit 7 low
ori   8Øh        ;preserve data but bring bit 7 high
out   base       ;output data with bit 7 high
ret
```

This routine would be appropriate for Centronics style printers expecting a strobe in data bit 7.

Caution!

Remember that the Group Select Port, BASE+7, has other functions besides selecting the current group. As described in this section, bringing bit 5 low disables the parallel output ports. Bank select, Interrupt Enable and printer Restore are also controlled by this port. Please read the appropriate sections of this manual.

4.2.2. THE DAISY PORT AND INTERRUPTS

The Print Wheel Ready status line of the DAISY port (P4 connector pin 27, BASE input port bit 5) is brought through an inverter to Interrupt Request line 6 of the 8259-A PIC. The PIC can therefore generate an interrupt whenever this line goes to an active (i.e. logic low) state. To take full advantage of this interrupt option when interfacing with a Daisy Wheel printer, and to exploit the Diablo printer's ability to buffer motion commands, printer driver software should be written so that the Print Wheel Strobe (P4 pin 21, BASE output port bit 6) is not activated until all carriage positioning commands have first been sent to the printer. Print after space will execute significantly faster than space after print. When the Print Wheel Ready line goes active the printer should be able to accept another motion-then-print sequence.

A sample Diablo printer driver for the MULT/IO can be found in the Appendix of this manual.

5. REAL TIME CLOCK: THE 199Ø

The 199Ø CMOS crystal-controlled calendar/clock chip at location at 15D supports a real-time environment by providing two functions:

1) a calendar clock accessible from software able to run on battery backup when the system is shut down;

2) a timed interrupt generator capable of providing real-time interval interrupts with three software programmable lengths.

The clock uses 6 bits of port BASE+2 for control and entering time. The time can also be read through this port. Inputting this port resets the timed pulse interrupt latch, which is connected to the lowest priority interrupt on the 8259 PIC.

5.1. 199Ø ARCHICTECTURE

The 199Ø Calendar/Clock chip maintains the time in an internal register. This register is loaded or read by sending a command to the chip which transfers the time information between the internal register and a shift register. The shift register is used to set or read the time, a bit at a time.

The time is stored as in Binary Coded Decimal (BCD) format. That is, each digit is represented as a 4 bit (one nibble) decimal digit between Ø and 9. The exception to this is the month nibble, which is stored as a hex digit between Ø and 11.

The clock automatically increments the minutes, every 6Ø seconds, hours every 6Ø minutes and days every 24 hours. Saturday, day 6, is followed by Sunday, day Ø. The hours are maintained in 24 hour notation (Ø hour to 23ØØ hours), and months are incremented after 31 days. Since every month is 31 days for the clock chip, software must be used to correct for shorter months.

Setting the time is done by shifting in 4Ø bits of information, using the Clk pin as a strobe, and then issuing a command to load the shift register into the internal register, using the STB bit as a strobe. Reading the time operates in reverse order. The section on programming the 199Ø gives more exact details. Also, there is a software example in the back of this manual.

5.1.1. THE CLOCK PORT

Seven pins of the 199Ø Calendar/Clock chip are connected to port BASE+2. One output bit is for data, two are for strobes and three are for control of the chip. Only one input bit is available, for reading the time. Reading and writing the clock is done by an internal shift register. The section on programming the 199Ø explains accessing the clock.

The charts that follow give a description of the correspondance between 1990 pin and data bits in the Clock Port, the meanings of the pins and the various control codes.

Table 5-1: 1990 CALENDAR/CLOCK I/O MAP

I/O Port BASE+2	BASE+2 Data Bit	1990 Pin # & Mnemonic	1990 Function
INPUT	0	9 Data out	Output of shift register
OUTPUT	0	6 Data in	Input to shift register
	1	8 Clk	Strobe for shift register
to	2	3 C0	Command bit 0
	3	2 C1	Command bit 1
1990	4	1 C2	Command bit 2
	5	4 Stb	Strobe for command

5.1.2. CLOCK COMMANDS

The 1990 clock has two sets of commands: the first, with C2 set to a 0, controls the shift register; the second, with C2 set to 1, sets the timed pulse or test mode. The table which follows describes the possible commands:

Table 5-2: CLOCK COMMANDS

Function	C2	C1	C0	
Shift register hold	0	0	0	Control
Enable shift register	0	0	1	Shift
Load clock from shift reg.	0	1	0	Register
Load shift reg. from clock	0	1	1	
TP = 64 Hz.	1	0	0	Set
TP = 256 Hz.	1	0	1	Timed
TP = 2048 Hz.	1	1	0	Pulse
Test Mode (32 Hz.)	1	1	1	

Commands to the 1990 must be strobed in, that is, the Stb pin, bit 5 of the clock port, must be changed from a zero to a one and back to a zero while the command remains unchanged. The transitionofthe Stb bit from high to low actually latches the command into the clock chip.

5.1.3. PROGRAMMING THE CLOCK: INTIALIZATION

When power is first applied to the clock chip, it goes into test mode. If a battery backup is used, it should remain in the last command mode issued. Before any shift register commands can be issued, one of the three timed pulse intervals (TP) must be selected. In fact, whenever Test Mode is entered, a TP interval must be selected before the clock will accept any shift register commands.

NOTE: The 199Ø Calendar/Clock chip always generates Timed Pulses. This is connected to IRQ 7, and will generate an interrupt unless it is masked in the 825Ø PIC, or interrupts are disabled.

To select a Timed Pulse interval, the three command bits are output with the strobe bit low, then high, then low again. The clock chip uses the low going edge of the strobe (Stb) to latch the command. To set the clock for 64 Hz. Timed Pulse interval, the following sequence should be followed:

Set the Stb, CØ and C1 bits to Ø, and the C2 bit to a 1 and output to BASE+2 of group Ø;

Set the Stb bit to a 1, and output the command again, with the other bits unchanged;

Set the Stb bit to a Ø without changing the other bits and output it.

Once this has been done, the clock will accept shift register commands. Shift register commands are read by the clock in the same manner, that is, each command is issued with the Stb bit low, then high, and then low again.

5.1.4. PROGRAMMING THE 199Ø CLOCK: SETTING THE TIME

The 199Ø time is set by giving it a shift register command, shifting in 4Ø bits of time and date, and issueing the load clock from shift register command. Bits are shifted into the shift register in a manner similar to strobing in the commands. Each data bit is output to the clock port with the Clk bit first set to Ø, then to 1, and back to Ø. Just as in the command sequence, it uses the Clk bit, bit 1 of BASE+2 (group Ø), to latch each data bit on the high going edge of Clk. When all 4Ø bits have been strobed in, the load clock from shift register command is strobed in using Stb.

The sequence for shifting bits is:

Strobe in shift register command;

Output first data bit with Clk set to Ø;

Output first data bit with Clk set to 1;

Output next data bit with Clk set to Ø;

Output same data bit with Clk set to 1;

Repeat the two previous steps until all 4Ø bits are shifted;

Output the last data bit with Clk set to Ø;

Strobe in the load-clock-from-shift-register command.

When setting or reading the clock, it is suggested that interrupts are disabled.

5.1.5. PROGRAMMING THE 1990: READING THE TIME

The time is read from the 1990 in much the same manner that it was set: the load shift register from clock command is strobed in; the shift register command is strobed in; the Clk bit is brought low, high, low to strobe the shift register; and the data bit is read on bit 0 of the clock port. One point should be noted: the first data bit is available before the shift register has been shifted and can be read immediately. This sequence is outlined below:

Strobe in load shift register from clock command;

Output Clk bit set to 0;

Strobe in shift register command;

Input data bit in bit 0 of port BASE+2 Group 0;

Output Clk bit set to 1;

Output Clk bit set to 0;

Repeat previous three steps until 40 bits have been read.

The format of the data bits shifted out of or into the shift register is described in the next section.

5.1.6. FORMAT OF THE 1990 TIME

The 1990 Clock/Calendar chip stores the time as 40 bits in a FIFO shift register. FIFO means that the first bit shifted in is the first bit shifted out. In the case of the 1990, the least significant bit (LSB) of the seconds units is shifted in first, and the most significant bit (MSB) of the month is shifted in last. In reading the clock, the same order is maintained, the first bit out being the LSB of the seconds units.

The format of the time is in BCD digits. For example, Thursday, the 29 of October, 1:08::50 P.M. is represented by:

Figure 5-1: TIME FORMAT EXAMPLE

Time	OCT	THUR	29		1:		08::		50	P.M.
Decimal	9	4	2	9	1	3	0	8	5	0
BCD	1001	0100	0010	1001	0001	0011	0000	1000	0101	0000
	MSB									LSB

There are several things to note in the format. First, the months November and December are entered as hex digits Ah and Bh. The first day of the week is Sunday (coded as Ø), and the last is Saturday (coded as 6). The clock keeps the hours in 24 hour notation: 1 o'clock is 13ØØ hours and 11 o'clock is stored as 23.

The following figure goes into more precise detail the format of the internal clock of the 199Ø:

Figure 5-2: 199Ø INTERNAL CLOCK FORMAT

NAME	MONTH	DAY	DATE/TENS	DATE/UNITS
BIT #	4Ø . . 37	36 . . 33	32 . . 29	28 . . 25
Shift IN->				
BCD	1 Ø Ø 1	Ø 1 Ø Ø	Ø Ø 1 Ø	1 Ø Ø 1
Example:	October	Thursday	2	9

NAME	HOUR/TENS	HOUR/UNITS	MINS/TENS	MINS/UNITS
BIT #	24 . . 21	2Ø . . 17	16 . . 13	12 . . 9
BCD	Ø Ø Ø 1	Ø Ø 1 1	Ø Ø Ø Ø	1 Ø Ø Ø
Example:	1	3 :	Ø	8

NAME	SECS/TENS	SECS/UNITS	
BIT #	8 . . 5	4 . . 1	--> Shift OUT
BCD	Ø 1 Ø 1	Ø Ø Ø Ø	
Example:	5	Ø	

Sunday = Ø	January = Ø	1:ØØ PM = 13
Monday = 1	February = 1	2:ØØ PM = 14
...	...	...
Saturday = 6	December = 11	12:ØØ PM = ØØ

When setting the time, bit 1, the LSB of the seconds/units, would be shifted in first, (a Ø in this case), and bit 4Ø, the MSB of the month, a 1, would be shifted in last. When reading the time, bit 1 would come out first (Ø), bit 4Ø last (1 in this example).

5.1.7. CALENDAR CLOCK IDIOSYNCRACIES

Once the 4Ø bit shift register of the 199Ø has been set with the desired date and time, and loaded into the internal register, it automatically increments the time and date for later references. Note, however, that the 199Ø considers all months to have 31 days, so September, April, June and November -- and of course February -- require a special update at the end of each month to keep the calendar current. The end of the year also requires a special update. After New Year's Eve, the clock wakes up quite confused about what day it is, and should be reloaded.

5.1.8. TIMING CONSTRAINTS

The 199Ø is not capable of reading or writing serial data fast enough to keep up with the CPU unless the Clk and Stb bits are prolonged for about 4Ø micro-seconds. The software routines descibed above accomplish this.

5.1.9. THE TIMED INTERRUPT GENERATOR

In addition to being a calendar/clock, the 199Ø is capable of generating interrupts at timed intervals. The interrupts generated by the 199Ø are routed to Interrupt Request number 7 of the 8259-A PIC. In order for these interrupts to be received properly, the PIC must be set to operate in the level, rather than the edge, mode. The 199Ø continously generates timed pulses, so it is important to mask out these interrupts if they are not desired. Three intervals are available:

1) Once every .488 milliseconds, or 2Ø48 interrupts per second

2) Once every 3.9 milliseconds, or 256 interrupts per second

3) Once every 15.Ø milliseconds, or 64 interrupts per second.

Please refer to the section on Clock Commands for setting the Timed Pulse intervals.

5.1.1Ø. CLEARING CLOCK INTERRUPTS

Any input instruction directed at I/O port BASE+2 of group Ø clears the interrupt request generated by the 199Ø. This action does not involve the 199Ø clock chip, but clears the flip-flop throughwhich the 199Ø TP output is latched and converted to a constant level before reaching the 8259-A PIC. The data obtained from this instruction should be ignored.

5.1.11. BATTERY BACKUP

Provision has been made for battery backup to the real time clock. By providing 3 volts to the 199Ø, the clock will continue running for approximately the shelf life of the battery. The 199Ø is a CMOS device which draws very little current. It also should be protected from voltages higher than 3.3 volts.

If a nickel-cadmium battery is used, it can be trickle recharged by installing a resister (18 kohm suggested). The trace to the left of the 199Ø should be cut, and the resistor installed.

The header at P6 (located at D14) is used to supply backup power to the 199Ø. The middle pin is for +3 volts, and the two side pins are connected to ground.

Please refer to the 199Ø specifications for more information on battery backup details.

6. INTERRUPT SYSTEMS

Microcomputer systems in general are required to communicate with peripheral devices such as printers, CRT terminals and various types of parallel devices. There are classically two ways of approaching the way a CPU may service these devices - polled and interrupt.

In a polled mode, every device in the system is periodically querried about its service requirements. When a device requires servicing (for example, a person has just typed a charactor on a CRT terminal), the CPU stops polling all other devices until it has finished servicing the user's request. Often times, a device must be serviced within a critical time period, or data is lost. Serialdata from a modem would be an example of this: if a charactor is not read before the next charactor is received, it is lost.

From a system viewpoint, the CPU should handle these requests as quckly as possible. The total system throughput is a function of the number of devices on the system, the length of time to poll each device and service each device request. The operating system is continually polling the devices searching for activity, even in the midst of other tasks. This reduces the time available for actual computation.

There is a direct analogy here to hardware design: this type of operation is said to synchronous. This means that the CPU may branch to a device service routine only after it has determined through polling that it is necessary to do so. The timing is then dependent on the program controlling the CPU, rather on the timing of the device that requires servicing.

There is another problem with this approach. This lies in the lack of priority setting. In a polled system, each device has equal status, which is unfortunate because in a real environment some devices require faster, more frequent servicing than others. Polling high priority devices more frequently is one solution, but this burdens I/O subroutines with complex algorithms.

An interrupt-driven system is much different in its implementation. Although requiring more hardware and more difficult to design software, the system has none of the problems associatedwith polled systems. With correct hardware, the devices are all prioritized according to their service requirements and the CPU is free to handle other tasks until a device requires service.

The I/O devices in this system interrupt the CPU only when they require something from the host processor. This type of system is more analogous to an asynchronous hardware design - one where events can occur at random intervals not related to the CPU's operations. Its randomness corresponds nicely with the relative randomness of device requirements tied into the system andallows maximum response to these peripherals.

6.1. INTERRUPT REQUIREMENTS

8Ø8Ø and Z-8Ø microprocessors monitor one control line, PINT/, and expect a particular sequence of events to follow a request on PINT/. This involves informing the CPU what section of memory contains the code for the interrupt service routine.

When the processor receives the PINT/ signal, it completes its current instruction, then issues a signal called INTA/. INTA/ is the interrupt acknowledge signal. When it is asserted, the CPU expects to receive its next instruction from the interrupting device. The CPU's Program Counter is not incremented during INTA/. Asserting INTA/ will also usually disable memory so that the address lines will be ignored by most memory boards.

At this point, a device may generate any instruction it wishes and the host CPU will execute it. Two instructions most probably will be asked of the CPU in such a case - a Restart or a Call. These are logical choices because both of them predictably alter the current flow of instructions by changing the Program Counter to a particular address, and saving the old Program Counter by pushing it on the stack. A Restart instruction is limited to eight locations in memory, and may interfere with other software that uses the Restart locations. This leaves us with the Call instruction.

There are some differences between the Z-8Ø and the 8Ø8Ø in their response to PINT/. The 8Ø8Ø will generate, through an intermediate device, interrupt acknowledge for the next three memory reads. The Z-8Ø only issues one. This difference is resolved by integrated logic on the MULT/IO board, which issues address disable for two cycles, which prevents the Z-8Ø from driving the address lines. PHANTOM/ will be made true during these two cycles by jumpering the pins at J6.

The Z-8Ø also has three interrupt modes, so that it must be set in in Interrupt Mode Ø, and an EI, enable interrupt must be executed.

6.2. PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

The programmable interrupt controller, in conjunction with standard integrated circuits, provides the hardware requirements for Z-8Ø and 8Ø8Ø interrupt systems. The 8259-A PIC can directly monitor eight devices and prioritize them according to system requirements. It issues Call instructions in response to INTA/ and also provides addresses for eight different interrupt routines.

Program controlled functions allow the system designer flexibility in designing the operating system. Priorities may either remain fixed or rotate automatically, or rotate under program control. The addresses for interrupt service routines may be assigned to anyplace in memory. The 8259-A PIC may also be "slaved" to a "master" PIC, allowing up to four MULT/IO's in the same system.

6.2.1. PIC INTERRUPT VECTORS

The PIC is designed to generate a Call Instruction upon receiving the INTA/ response from the host CPU. The CPU then expects a 16 bit address of the location of the interrupt vector. Hardware on the MULT/IO counts the next two CPU fetches (for the address vector) and enables the PIC to put this address on the data-in bus. When programmed, the PIC has eight addresses associated with the eight devices it monitors. These addresses hold the jump instruction to the service routine for each device.

The PIC generates interrupt vectors at either eight-byte or four-byte intervals in the 16 bit address space, limited by both the PIC and the CPU to a 64K address space. For compactness, most systems use the four-byte interval since a jump instruction is only three bytes long. It would be very difficult to have an interupt routine in eight bytes. The eight byte interval was provided for compatibility with the Restart instruction locations, which are spread eight bytes apart.

Five of the PIC's Interrupt Request lines (IRQØ-7) are hardwired to devices onthe MULT/IO board. These are the three serial devices, pin 5 of the parallel input port (DAISY print wheel ready) and the timed pulse line of the real/time clock. The other three IRQ's are jumpered to the first three vectored interrupt lines of the S-1ØØ buss: VIØ, VIl and VI2. These may be changed by cutting the jumpers and installing new ones to other VI lines.

Table 6-1: MULT/IO CONNECTIONS TO THE PIC INTERRUPT REQUEST LINES

Priority	Interrupt Request Line	MULT/IO Device/Connection
Highest	IRQØ	S-1ØØ vectored interrupt Ø
	IRQ1	S-1ØØ vectored interrupt 1
	IRQ2	S-1ØØ vectored interrupt 2
	IRQ3	ACE #1 (serial device)
	IRQ4	ACE #2
	IRQ5	ACE #3
	IRQ6	DAISY print wheel ready
Lowest	IRQ7	Timed pulse from clock

The priority assignments in this table are for the nested mode of the PIC, and may be varied by programming to different priority. The order remains the same.

6.3. PRIORITY MODES OF THE PIC

Much of the flexibility of the 8259-A PIC is in its array of priority modes. Through the initialization and operation control words, the system designer can choose between fixed priorities and software variable priorities.

Interrupt priorities are important because they allow servicing of time-critical devices ahead of devices with less demanding constraints. In some cases, it may be necessary to allow lower priority devices to interrupt a service routine after itstime-critical section has been completed. The various priority modes described in this section provide the programmer with solutions to a wide variety of priority requirements.

In general, a device with a priority less than or equal to a device which has an interrupt in progress will not be allowed to interrupt. When the higher priority device's service routine signals its end of interrupt, the lower (or equal) priority device will be able to interrupt. Thus, higher priority devices can lock-out devices with lower priority.

The end of interrupt (EOI) is the command sent to the PIC to signal that a device's service routine is finished with its time-critical portion, and lower priority interrupts can be enabled. The EOI command and its variants will be explained in greater detail later.

The priority assignment modes of the 8259-A PIC will be described in increasing order of complexity. The first mode, the polled mode, does not use the interrupt capability of the PIC, which must be disabled by changing jumpers on the MULT/IO board. All other modes use the interrupt capabilities of the PIC. The differences between them are in the manner in which the priorities are maintained in the PIC.

6.3.1. POLLED MODE

The PIC may be configured to resemble a polled I/O system by setting the polled mode bit. Interrupts must be disabled in this mode. The PIC may generate an interrupt in this mode with a change in state of any of its IRQ lines. To prevent interrupt requests from the MULT/IO, the jumper between B of J5 and PINT/ should not be connected. (The section on configuring interrupt jumpers gives a more detailed explanation of this.) The CPUmust poll the PIC to see if any device is requesting service. If a device is requesting service, the most significant bit is set to 1, and the highest priority device requesting service is encoded in the lowest three bits.

In polled mode, the command that enables polling must be output each time before the status information can be input.

6.3.2. NESTED MODE

The nested mode of the PIC allows service requests from I/O devices to be prioritized. When a device is in need of service, the PIC issues an interrupt to the CPU only if there are no higher priority devices requesting service via the PIC. If a lower priority device requests service, it must wait until all higher priority devices are serviced and the interrupt handling routine has issued an end of interrupt command to the PIC.

If a device with a higher priority requires service, the lower priority device's service routine is interrupted until the higher priority device has been serviced. However, before the lower priority device's service routine can be interrupted, an EI (enable interrupt) command must be issued. This mode provides maximum system response to devices which require immediate service. All Morrow Designs software takes advantage of the PIC nesting.

6.3.3. FULLY NESTED MODE

The fully nested mode is used when one PIC is used as a master to several slave PIC's and priority is to be maintained in each PIC. In other words, an interrupt request can be granted at the same priority. This means that when a slave PIC that has interrupted the master PIC has a higher priority interrupt pending, the higher priority interrupt (within the slave) will be serviced first.

In nested mode, the request from the slave is masked while the previous request is in service. In fully nested mode, the slave's higher priority request will be handled before the lower priority request at the same IRQ line. In this case, all of the slave's requests must be serviced before an end of interrupt is sent to the master PIC.

6.3.4. ROTATING PRIORITY - MODE A

In the nested mode, devices are prioritized and the devicewith the highest priority obtains service. The priorities are assigned according to which request line (IRQØ-IRQ7) the device is connected. The scheme works well for devices not inherently equal. In some instances, all devices connected to the PIC should have the same priority.

The PIC may be programmed to rotate the priority through all devices. In this mode, each device gets rotated to the lowest priority after it has been serviced. The next device in order becomes the highest priority device. This prevents devices from "hogging" service when it should be evenly distributed.

6.3.5. ROTATING PRIORITY - MODE B

This mode is very similar to Mode A, the difference being that the rotation can programmed rather than fixed by hardware.

Instead of the priority being rotated so that the last serviced device is lowest, the device that has the lowest priority is selected by software. The device that would have the second lowest priority is now highest.

6.3.6. SPECIAL MASK MODE

The special mask mode is a way of temporarily altering the interrupt priority. By setting the special mask mode and altering the interrupt mask, devices of any priority may be serviced before the currently in-service interrupt has been ended. The mask is used to inhibit interrupts of specific levels, while enabling all others. This allows lower priority devices to be serviced before a higher priority device has issued an end of interrupt command. The interrupt service routine that invokes the special mask mode should also return the interrupt mask to its previous state before ending.

6.4. PIC STATUS REGISTERS

The PIC status registers may be read to determine the current state of the PIC. These registers place IRQØ - IRQ7 status on the data-in bits, Ø - 7 respectively. IRQØ is the highest priority and IRQ7 the lowest in nested mode. Accessing of these registers is explained in the next chapter.

6.4.1. INTERRUPT MASK REGISTER (IMR)

The PIC has the capability of masking any of the eight interrupt inputs - that is, not allowing that device to generate an interrupt. The mask register contains eight bits, any of which, when high, shut off the appropriate IRQ input to the PIC. If all the bits are set high, no interrupts are generated. If all are set low, all devices are recognized in their normal prioritized sequence. This allows the software complete control over each individual device's service requests.

This register can both be read and written to by system software. It is also called Operation Control Word 1 (OCW1).

6.4.2. IN-SERVICE REGISTER (ISR)

The in-service register allows the software to query the PIC about which devices are currently in service. Anytime an interrupt is generated by the PIC, the bit corresponding to the request line granted the interrupt is set. Thus, any interrupt routine currently in progress, and any routine that was interrupted by a higher priority routine, will have a bit set high. These bits are reset by an end of interrupt command issued by the associated interrupt service routine.

6.4.3. INTERRUPT REQUEST REGISTER (IRR)

This eight-bit register is read to determine which of the eight devices is requesting service. The highest pending priority is reset whenever an interrupt from the PIC has been acknowledged by the CPU (INTA/ issued). Bits representing still pending interrupt requests stay high (set to 1).

6.5. OTHER PROGRAMMABLE FEATURES

The 8259-A has other software programmable features besides arranging interrupt priorities. These are directly related to hardware design and the number of PIC's in use. Some of the hardware related features are mainly dictated by design, such as Buffered Mode and Level Triggered Mode. The implementation of multiple PIC's on several MULT/IO's is also affected by the design of the MULT/IO. The next sections explain these modes.

6.5.1. TRIGGERED MODES

The PIC may be programmed to monitor the eight request lines in either edge-triggeed or level-triggered mode (LTIM). In edge triggered mode, the PIC generates an interrupt after a high to low transition on the request lines (IRQØ - IRQ7). This is suitable for devices that do not latch their interrupt requests. However, this does cause a problem because UART's may only generate one edge for one or more interrupts. The result is the loss of some interrupt requests. For this reason, all Morrow Designs software uses the level-triggered mode.

6.5.2. BUFFERED MODE

The buffered mode allows the PIC to generate a buffer enable signal during interrupt acknowledge cycles. This signal is used only in multiple MULT/IO systems. When the PIC is programmed to be a slave, it places the two vector address bytes on the data-in bus during the second and third cycles of the interrupt acknowledge sequence. The buffered mode is used to enable the data-in buffers on the slave MULT/IO during this sequence.

6.5.3. CALL ADDRESS INTERVAL (ADI)

The spacing between the call vectors for the interrupt service routines can be programmed at either four or eight byte intervals. Normally, four-byte intervals are used because jump instructions require three bytes. The eight byte interval is provided for compatibility with Restart instructions.

6.5.4. MICRO-PROCESSOR MODE

This mode allows the use of an 8Ø86 microprocessor. The 8Ø86 expects a two byte interrupt acknowledge sequence, as opposed to the three byte sequence of the Z-8Ø and the 8Ø8Ø/8Ø85. When this modeis selected, only the five most significant bits of the interrupt control vector are sent as the second byte of the interrupt acknowlegement. This feature is not used by Morrow Designs software. It is also not compatible with Z-8Ø Interrupt Mode 2.

6.6. CASCADING PIC'S

More than one MULT/IO may be used in the same system by cascading the 8259-A PIC's. When this is done, one PIC is the master. It controls the PINT/ line to the CPU and acknowledges the slave interrupt requests through the cascade lines. The other PIC's are configured as slaves. Their interrupt lines are connected to the Vectored Interrupt lines (VIØ - VI7) and their requests are mediated by the master PIC. The architecture of the MULT/IO allows up to three slaves in interrupt mode.

The cascade lines are outputs on the master PIC and inputs to the slaves. During an interrupt acknowledge sequence, the master PIC places the Call instruction on the data-in bus and signals the slave, by driving the cascade lines, to place the vector address on the data-in bus during the second and third INTA/ cycle.

When the master is initialized, a control word is issued that tells it which IRQ lines are connected to slaves. When a request is received on one of these lines, the master asserts the BCD code of the request line on the cascade lines. Each slave must be initialized with the code of the request line it is connected to on the master. In the case of the MULT/IO, the only possible code for slaves is Ø, 1 or 2, because these are the request lines that may be used.

6.6.1. MASTER/SLAVE MODE

At the beginning of initialization, a bit is set (SNGL)that establishes whether there are one or more PIC's in the system. When there is only one, SNGL is set to a 1. When there are more than one SNGL is set to Ø and an additional two control words must be issued for initialization. The first word is for control of the cascade lines, which is described above. The second word is used to establish whether this PIC is a master or a slave.

6.7. AUTOMATIC END OF INTERRUPT MODE

The automatic end of interrupt mode (AEOI) allows the PIC to clear the most recent in-service bit. Normally, the interrupt service routine must send an end of interrupt control word to the PICto clear the in-service bit and allow lower or same priority interrupts. Setting the AEOI bit to a 1 automatically clears the highest priority in-service bit at the end of an interrupt acknowledge sequence (INTA/). This allows other interrupt requests to be serviced immediately if the CPU has had its interrupt enable flip-flop reset (EI instruction).

7. PROGRAMMING THE 8259-A PIC

Before the 8259-A PIC can be used, it must initialized with at least two control words. If the operating system is not using interrupts, it is a good idea to set up the PIC or physically disable its connection to PINT/.

Bit 3 of the Group Select Port is used to enable/disable interrupts. Whenever a byte is output to port BASE+7 with bit 3 set to a 1, interrupts are enabled. If this bit is reset (Ø), interrupt requests will never reach the bus.

All of the flexibility of the 8259-A is programmed by theoutput of Initialization Control Words (ICW) and Operation Control Words (OCW). The PIC has only two ports associated with it, but uses seven control registers and four status registers. To access this multitude of registers, the correct sequence of control words must be adhered to.

The Initialization Control Words are issued whenever the system is reset or powered up. At least the first two ICW's must be issued at this time. Any time after initialization Operation Control Words may be issued. These are used for active control of the prioritizing scheme within the PIC and for selecting which status register is read.

7.1. INITIALIZING THE PIC

The PIC is initialized by outputting the first Initialization Control Word, ICW1. ICW1 is issued by outputting a byte to port BASE+4 of Group Ø with bit 4 set to 1. Anytime a byte is output to this port with bit 4 set to 1, an initialization sequence begins. Once the sequence begins, port BASE+5 of Group Select Ø becomes ICW2. ICW2 always follows ICW1. It contains the high byte of the interrupt vector address and will always be used.

The next two ICW's, 3 and 4, will need to be intialized according to the bits set in the first ICW. They are also at BASE+5 of Group Ø. If there are more than one PIC in the system, SNGL will be set to Ø (false) and ICW3 will need to be output. If the bit named ICW4 is set to 1, then ICW4 will need be to programmed. This follows ICW2 in single PIC systems and ICW3 in multiple PIC systems.

7.1.1. INITIALIZATION CONTROL WORDS 1 AND 2

ICW1 always begins an initialization sequence. It controls the sequencing of registers at BASE+5. It is also used to set triggered modes and address interval, and to set the address lines A7, A6 and A5 of the low byte of the vector address.

ICW2 always follows ICW1. It contains the high byte of the vector address. Bit 7 of ICW2 corresponds to A15 of the vector address and bit Ø to A8. During an interrupt acknowledge sequence, the PIC will enable two bytes onto the data-in bus. The first byte is the low byte of the vector address. This byte is made from the address bits A7-A5 of ICW1 and an offset. This offsetis determined by which interrupt request has been granted and the interval selected between address.

The offset is determined by multiplying the interrupt request line number by the address interval (IRQn x ADI). For example, if the ADI is four-bytes and the interrupt request being acknowledged is IRQ4, the offset will be 16. If the ADI were eight bytes and the IRQ was 7, the offset would be 56. When an ADI of eight bytes is used, address bit 5 of ICW1 is determined by the offset, because AØ to A5 are used to represent addresses in the range of Ø to 63.

Interrupt vector addresses will always be on 32 or 64 byte boundaries. This is because the offset explained above will set low bits of the first vector address to all zeroes.

EXAMPLE: The memory between 24ØØH and 242ØH has been set aside for the jump table to interrupt service routines, with the ADI set to four bytes. IRQØ is attached to a disk controller interrupt line through VIØ, IRQ1 and IRQ2 are not used, and the other IRQ lines are connected to the MULT/IO devices. The jump table would look like this:

Table 7-1: EXAMPLE OF JUMP TABLES FOR SERVICE ROUTINES

Address	Instruction	IRQ line	Device
24ØØ	JUMP DSKSER	Ø	Disk controller
24Ø4	JUMP SURPZ1	1	No connection
24Ø8	JUMP SURPZ2	2	No connection
24ØC	JUMP SERDV1	3	ACE #1
241Ø	JUMP SERDV2	4	ACE #2
2414	JUMP SERDV3	5	ACE #3
2418	JUMP PRNTWH	6	Parallel printer
241C	JUMP TIMOUT	7	Real/time clock

Each JUMP is to a service routine, except SURPZ1 and SURPZ2 which should never occur. Note that a JUMP instruction is only three bytes long, so that a byte must be inserted after each JUMP address. Also, remember that if an area outside of the program area is selected for this jump table, these instructions must be written before the PIC is initialized.

ICW1 controls the sequence of initialization. ICW2 always follows ICW1. If bit 1 (SNGL) is set to a Ø, then ICW3 follows ICW2. When bit Ø (ICW4) is set to 1, then ICW4 will be the last Initialization Control Word.

ICW1 has three other functions. Bits 7, 6 and 5 are used to set A7, A6 and A5 of the vector address. The interval between vector address is set by ADI, bit 2. When ADI is set to 1 then address interval is four-bytes; when it is Ø, the interval is eight-bytes. (The four-byte interval is normal for Morrow Designs software.) Bit 3 is LTIM. This is used to choose between edge and level triggered modes. Level triggered mode (LTIM = 1) is used on the MULT/IO.

ICW2 is output to BASE+5 of Group Ø immediately after ICW1. ICW2 contains the high byte of the interrupt vector address. This word must always be output, even if polling mode is to be used. The following tables recaps the bit assignments of ICW1 and ICW2.

Table 7-2: INITIALIZATION CONTROL WORD 1
PORT BASE+4 (whenever bit 4 = 1)

Bit	Name	Function
7	A7	These bits make up three msb of low byte of interrupt vector address.
6	A6	
5	A5	
4	ICW1	Set to 1 to signify beginning of initialization.
3	LTIM	Set to 1 for level triggered mode (Ø = edge).
2	ADI	Four-byte interval if 1, eight-byte if Ø.
1	SNGL	Set to 1 for single PIC, Ø for multiple PIC's.
Ø	ICW4	Set to 1 allows access to ICW4. If set to Ø, PIC is initialized as master, non-buffered mode, no AEOI and in nested mode.

Table 7-3: INITIALIZATION CONTROL WORD 2
PORT BASE+5 (Immediately after ICW1)

Address Bit	15	14	13	12	11	1Ø	9	8
Bit of ICW2	7	6	5	4	3	2	1	Ø

7.1.2. INITIALIZATION CONTROL WORD 3 (ICW3)

This word is used for cascading several MULT/IO PIC's together. Its purpose is to identify which interrupt request lines (IRQ) have slaves attached when initializing a master. When initializing a slave, this is used to program the slaves identity number. The slave's identity is the binary representation of the IRQ line it is attached to. Only IRQØ, IRQ1 or IRQ2 are available on the MULT/IO, so the only slave identities are Ø, 1, or 2.

ICW3 is output to BASE+5 after ICW2 if SNGL of ICW1 was set to Ø. The table that follows outlines the bit assignments.

Table 7-4: INITIALIZATION CONTROL WORD 3

BIT #	7	6	5	4	3	2	1	Ø
MASTER	X	X	X	X	X	IRQ2	IRQ1	IRQØ
SLAVE	X	X	X	X	X	X	n1	nØ

IRQ2, IRQ1 or IRQØ are set to a 1 if there is a slave attached, or a zero if there is no slave attached.
X means this bit is not used on MULT/IO. n1 and nØ are the binary equivalent of Ø, 1 or 2, depending on which IRQ the slave is attached to.

7.1.3. INITIALIZATION CONTROL WORD 4 (ICW4)

This word is output to BASE+5 of Group Ø whenever ICW4 of ICW1 was set to a 1. It follows ICW3 when SNGL = Ø, or ICW2 when SNGL is true. If this word is not output, all its bits are cleared (set to Ø's). This word should always be initialized.

Only bits Ø - 4 are used in ICW4. Bit Ø is micro-processor mode. If the MULT/IO is used with an 8Ø86 processor, this bit is set to 1. Bit 1 is used to set the automatic end of interrupt mode. When this bit is a 1, the in-service bit of the ISR is cleared at the end of the interrupt acknowledge sequence.

Bits 2 and 3 work together. If bit 3 is a 1, the PIC is a Master when bit 2 is set to 1. When bit 3 is 1 and bit 2 is set to a Ø the PIC is a Slave. If bit 3 is set to a Ø, pin 16 becomes an input. This is not supported in the MULT/IO.

Bit 4 of ICW4 is used to select the fully nested mode. When this bt is set to a 1, interrupts of the same priority as a request already in-service are allowed. This fully nested mode is used when a PIC is Master to eight Slaves. This mode is not used in Morow Designs software. The table that follows recaps the bit assinments of ICW4:

Table 7-5: INITIALIZATION CONTROL WORD 4 (ICW4)

Bit	Function
7	Not used
6	Not used
5	Not used
4	Set to 1 to select fully nested mode
3	Set to 1 to select Master/Slave
2	Set to 1 for Master, set to Ø for Slave
1	Set to 1 to select AEOI
Ø	Set to Ø for 8Ø/85, Z-8Ø mode, 1 for 8Ø86 mode

7.2. OPERATION CONTROL REGISTERS

Once the PIC is initialized, it is ready to function as the system interrupt controller. Further changes in the PIC operating parameters are accomplished by programming a set of registers referred to as Operation Control Registers. These registers are used to affect the priority of interrupt requests and to issue end of interrupt (EOI) commands to the PIC.

7.2.1. OPERATION CONTROL WORD 1 (OCW1)

This is the mask register of the PIC. Setting bits to 1 in this register "mask out" corresponding interrupt requests. This register may be input or output at any time after initialization at BASE+5 of Group Ø.

Setting any of the bits high forces the PIC to ignore the interrupt request line associated with that bit. The bits are arranged with bit 7 corresponding to IRQ7 and bit Ø to IRQØ. The PIC clears this register to all Ø's (all interrupt requests enabled) on power up. It is a good practice to set this register after initialization.

7.2.2. OPERATION CONTROL WORD 2 (OCW2)

Thisregister is selected at BASE+4 of Group Ø whenever a word is output to this port with bits 3 and 4 set to Ø. This word is used to signal end of interrupt (EOI). It is also used for sending specific end of interrupt and for using the rotating priority modes.

Every interrupt service routines sends an end of interrupt to the PIC. This command clears the appropriate bit in the in-service register (ISR) allowing same or lower priority interrupts to occur. The non-specific EOI clears the in-service bit with highest priority. This is used for clearing the PIC of interrupts while in nested mode.

When using other modes, such as rotating or special masked mode, the specific end of interrupt must be used. Both of these modes allow the dynamic alteration of priority levels. When the service routine clears its in-service bit, it sends the BCD code of its IRQ in the lowest three bits along with the specific EOI. This is no more complicated than using the non-specific EOI, since each routine services a particular IRQ.

Rotating priority modes A and B are set using OCW2. These are both previously described in the section on operating modes of the PIC. The table that follows describes the bit assignments of OCW2:

Table 7-6: OPERATION CONTROL WORD 2

Bit	Name	Function
Ø	LØ	These three bits are used for specific EOI or in rotate mode B
1	L1	
2	L2	
3	OCW2=Ø	Both of these bits must be zero to access OCW2
4	OCW2=Ø	
5		These three bits are decoded to determine which command is being transmitted
6		
7		

Table 7-7: OCW2 COMMANDS (BITS 5 - 7)

Function	Bit-5	Bit-6	Bit-7
Clear rotate - Mode A	Ø	Ø	Ø
End of Interrupt (EOI)	1	Ø	Ø
Specific EOI (use LØ, L1, L2)	1	1	Ø
Set rotate - Mode A	Ø	Ø	1
EOI causes rotate - Mode A	1	Ø	1
Set rotate - Mode B	Ø	1	1
EOI causes rotate - Mode B (use LØ, L1, L2)	1	1	1

EXAMPLE: In nested mode, a service routine that has completed its critical section and wants to enable interrupts would output a 2ØH to port BASE+4 of Group Ø.

In special mask mode, a routine servicing IRQ 5 and is ready to enable lower priority interrupts would send 65H to port BASE+4 of Group Ø for a specific EOI of IRQ5.

7.2.3. OPERATION CONTROL WORD 3 (OCW3)

Operation Control Word 3 is used to further extend the flexiblity of controlling the PIC. It is used to access the polling register, the ISR and IRR registers, and to use the special mask mode. OCW3 is selected by outptting to port BASE+4 with bit 4 set to Ø and bit 3 set to 1.

NOTE: Three different control words use BASE+4: ICW1, OCW2 and OCW3. Whenever bit 4 of BASE+4 is set to a 1, an initialization sequence begins. Whenever bit 4 is set to Ø, OCW2 is selected when bit 3 is a Ø, and OCW3 is accessed when bit 3 is set to a 1.

After initialization theInterrupt Request Register (IRR) is accessed by reading port BASE+4 of Group Ø. The In-Service Register (ISR) can also be accessed at this port by using OCW3. Bits Ø and 1 are used to select which register is accessed at BASE+5. Whenever the selection is made, it remains the same until a different register is selected through OCW3.

When bit 1 (SRIS) is set, the register accessed will be IRR if bit Ø is a Ø and ISR when bit Ø is set to a 1.

OCW3 is also used to select the polled mode. Whenever a CH, that is bit 2 and 3 set to a 1, is output to BASE+4, the NEXT input from BASE+5 of Group Ø will be the BCD code (binary representation) of the highest priority interrupt pending. Every time the PIC is polled, an OCW3 with bits 2 and 3 set must be output just previous.

Special mask mode is also selected using OCW3. When both bits 5 and 6 are set high, special mask mode is selected. This allows use of the mask register to mask out selected requests AND enables lower and same priority requests. To deselect this mode, output OCW3 with both these bits reset (set to Ø).

The following table recaps the bit assignments of OCW3:

Table 7-8: OPERATION CONTROL WORD 3 (OCW3)

Bit	Function
7	Not used
6	ESSM - Enable Special Mask Mode when 1
5	SMM - Also must be set to 1 for SMM
4	Always Ø for accessing OCW3
3	Always 1 for accessing OCW3
2	Enter Poll mode on NEXT input of BASE+5 when 1
1	SRIS - Enable selection of IRR or IRS when 1
Ø	RIS - Selects IRR at BASE+5 when Ø, ISR when 1

7.3. SERVICE ROUTINE REQUIREMENTS

The following steps are necessary for any interrupt service routine working with the 8259-A PIC. In order to start up the interupt system, the operating system initializes the PIC and sends it operation control words if necessary, enables interrupts in the CPU (EI instruction), and gives an EOI command. Bit 3 of the Group Select Port is set to a 1 to enable interrupts. Then:

When the interrupt occurs, the ISR (interrupt service routine) saves the registers to be restored when control is returned to the interrupted routine; since an ISR may occur at anytime, no registers can be changed; the Group select port must also be returned to its previous state before exiting this routine;

Service the device which generated the interrupt;

Send an EOI command to the PIC; this allows the lower or same priority devices to be granted interrupt requests;

Restore all registers and the group select port to their state upon entry;

Enable interrupts (EI) in the CPU; this is necessary because the CPU automatically disables interrupts whenever an interrupt has been acknowledged;

Return to the interrupted program by issueing an RET command.

The EOI and EI commands may be issued before the ISR is completed if other lower or same priority interrupts are to be acknowledged.

Normally, registers are saved by pushing them on the stack. All the registers used in an ISR must be preserved in this manner. Before exiting the ISR, the registers are popped off the stack in reverse order.

Since it is necessary to change the Group Select port while servicing a device, this must also be restored before exiting the ISR.

8. CONFIGURING THE MULT/IO FOR THE PIC

Before the PIC can be used to generate interrupts, at least two jumpers must be installed on the MULT/IO board. When the MUL/IO contains the only interrupt generating devices in the system, only two jumpers must be made. At J4, located to the left of the LSØ4 at 12C, the slide-on jumper is used to connect pins B and C. At J5, located below 3D, pad B is connected to PINT/ with a wire to enable interrupts onto the S-1ØØ bus.

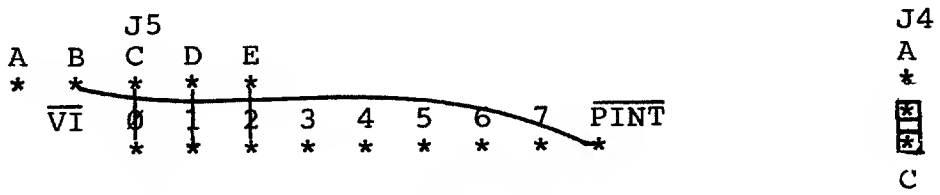


Figure 8-1: JUMPER AREAS J4 AND J5

Pads C, D and E are already connected to VI Ø, 1 and 2.

If the PIC is used to monitor other vectored interrupt lines than VIØ, VI1 and VI2, then the trace between these pads and pads C, D and E of J5 must be cut and a new wire installed.

There are three other modes that affect the configuration of the MULT/IO: polled mode, PIC as Master and PIC as Slave.

8.1. PIC IN POLLED MODE

If the PIC is to be used in polled mode, it will continue to generate interrupt requests at its interrupt pin. By leaving B of J5 unconnected, these requests will never reach the S-1ØØ bus, or the CPU. The slide-on connector at J4 should be used to connect together pins A and B. Connecting together these pins prevents the PIC from gating an address vector onto the bus during an interrupt acknowledge cycle (INTA/).

Making these changes physically disables the PIC's interrupt capabilities.

8.2. PIC AS MASTER

When the PIC is configured as Master, the physical connections are the same as when it is configured as a single board. If slaves are connected to vectored interrupt lines other than VIØ, VI1 or VI2, then the jumper between pads C, D and E of J5 and the vectored interrupt lines must be changed to correspond to the vectored interrupt lines used. The cascade lines must also be connected.

8.3. PIC AS SLAVE

The PIC configured as Slave has much the same connections as the Master. The only differance is that instead of connecting pad B of J5 to PINT/, pad B is connected to the vectored interrupt line

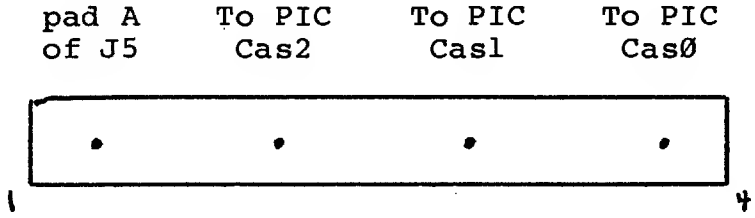
that the Master PIC is monitoring for that slave. Slaves are not allowedto generate PINT/. The Master receives the request from the Slave and issues PINT/ according to the priority of the Slave's request. The cascade lines must also be connected.

8.3.1. CASCADE CABLE

The cascade cable is used by the Master to communicate with its Slaves. During an interrupt acknowledge sequence (INTA/), the master enables a CALL instruction on the data-in bus. Then it uses the cascade lines to command the Slave to put the vector address of the service routine on the bus during the next two cycles. The Master knows a Slave should do this because a bit corresponding to the IRQ line the Slave is attached to was set in ICW3. And the Slave recognizes the code on the cascade lines because it was programmed with the BCD code of the IRQ line it is connected to with ICW3.

The cascade cable is connected to P5, located between the 8259-A PIC and P4, the parallel connector. It consists of four pins. One pin is connected to pad A in the J5 area. It may be used to route an interrupt request from a Slave to a IRQ line (pads C, D or E) if the other VI lines are all used. The other three pins are the cascades lines. The figure below illustrates P5, the cascade cable connection:

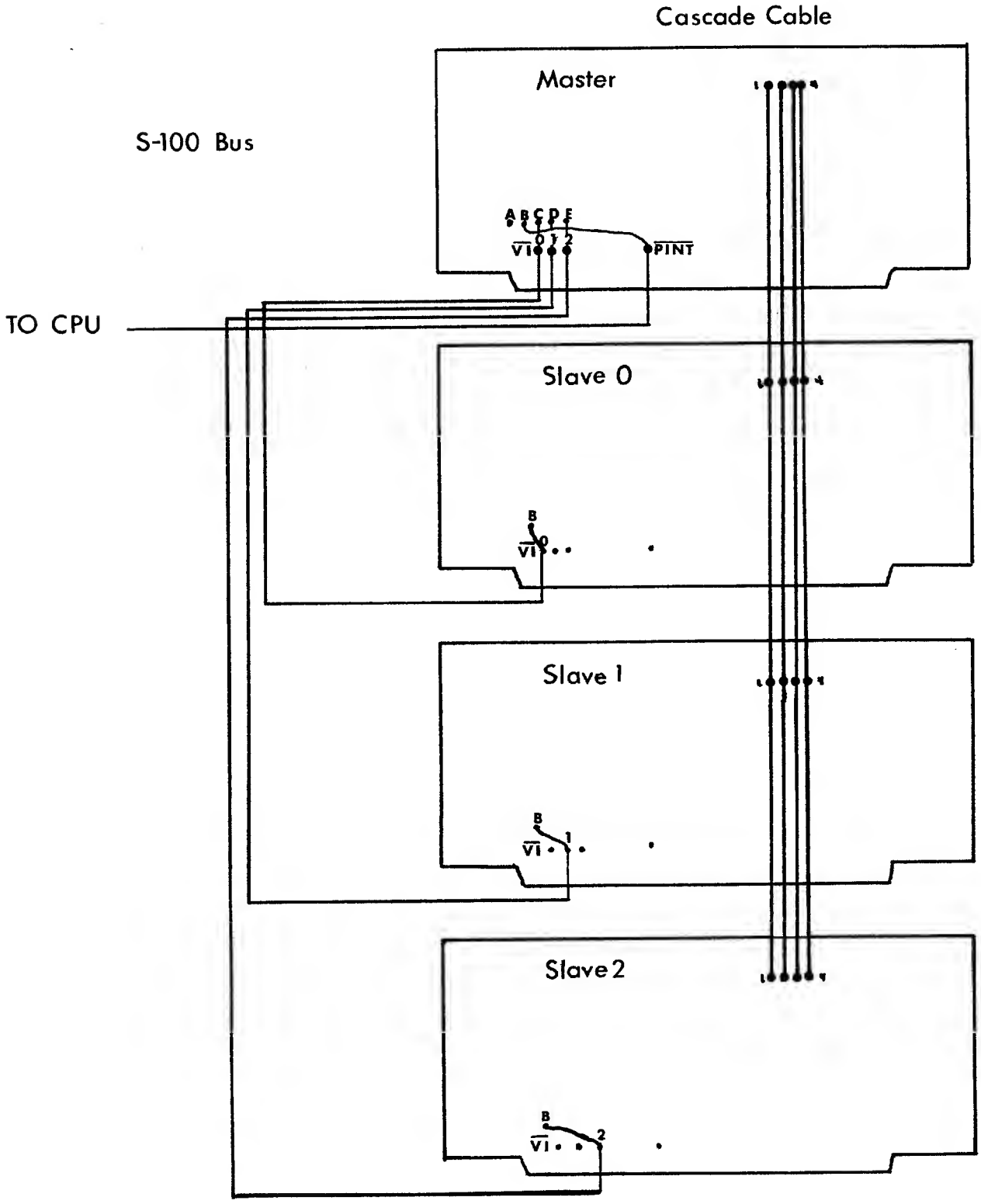
Figure 8-2: CASCADE CABLE CONNECTIONS (P5)



The BCD code that activates the Slave is asserted on the Cascade Lines by the Master during the second and third interrupt acknowledge cycles. The cascade cable can be made using three wires connected in the same order on each MULT/IO board. The leftmost pin is not essential. Even though the Master PIC will never bring Cas2 high (it only monitors IRQØ, IRQ1 and IRQ2 for exernal requests), it should be connected and not allowed to float.

The illustration on the following page shows the PIC on a MULT/IO as Master connected to three Slaves on other MULT/IO boards.

Figure 8-3: FOUR MULT/IO BOARDS IN MASTER/SLAVE CONFIGURATION



Software Samples

The following program tests the PIC's ability to co-ordinate interrupts generated from the bus vectored interrupt lines 0-2, from ACE serial device # 1 (controlling connector J1), and from the MULT/IO's clock/calender. The program assumes a working CP/M system with a terminal already interfaced and working. It also assumes a MULT/IO board addressed to begin at I/O port 48H in a system with no other enabled interrupt controller. The PIC is jumpered as the Master Controller (see section on configuring the PIC).

This program should cause the CP/M terminal device to print continuous asterisks (ASCII 2Ah), punctuated every second by an exclamation point caused by the clock's TP interrupt line. Grounding one of the first three vectored interrupt lines (V0-V2) will cause a message identifying that line. A terminal attached to connector J1 should meanwhile echo any character typed on it. The terminal attached to J1 should be set for 9600 baud with an 8 bit word length and two stop bits.

This program should be exited via a system reset. Notice that the routines START and SLOOP dynamically allocate the 32 byte vectored interrupt table (TABLE) to begin at an even 32 byte boundary. The PIC will only issue ISR CALL's to a table beginning at an even 32 byte boundary, and this is one of many ways to deal with this characteristic of the PIC.

```
000D =          ACR     EQU     0DH     ;carriage return
000A =          ALF     EQU     0AH     ;line feed
                ;
0048 =          BASE    EQU     48H     ;mult/io i/o base
004F =          GRPSEL  EQU     BASE+7  ;group select port
004C =          PIC0    EQU     BASE+4  ;pic port a=0
004D =          PIC1    EQU     BASE+5  ;pic port a=1
                ;
0010 =          D4      EQU     10H     ;bit to signify pic init command
0008 =          LTIM    EQU     8       ;level trigger interrupt mode
0004 =          ADDI    EQU     4       ;call address interval is 4
0002 =          SNGL    EQU     2       ;only one pic in system
0040 =          IMASK   EQU     40H     ;interrupt mask-- no p.w rdy
0020 =          EOI     EQU     20H     ;end of interrupt command to pic

                ;
0080 =          DLAB    EQU     80H     ;divisor latch access bit
0001 =          WLS0    EQU     1       ;word length select bit 0
0002 =          WLS1    EQU     2       ;word lenngth select bit 1
0004 =          STB     EQU     4       ;stop bit code for 2 stop bits
000C =          BRATE   EQU     12      ;baud rate constant for 9600
0002 =          ETBEI   EQU     2       ;enable tbe interrupt
0001 =          ERBFI   EQU     1       ;enable dav interrupt
0001 =          INTPEND EQU     1       ;interrupt pending status, 0 if pending
0006 =          INTTYP  EQU     6       ;int. id bits 1 and 2-- 0 means bad int
                ;
004A =          CLK     EQU     BASE+2
004B =          CLRCLK  EQU     BASE+3  ;port to clear clock interrupt
                ;
0014 =          TP256   EQU     14H     ;256 Hz tp pulse
0020 =          CSTB    EQU     20H     ;clock command strobe
                ;
0049 =          IER     EQU     BASE+1  ;interrupt enable register
004A =          IIR     EQU     BASE+2  ;interrupt id register
004B =          LCR     EQU     BASE+3  ;line control register
0048 =          DLL     EQU     BASE    ;least significant baud rate byte
0049 =          DLM     EQU     BASE+1  ;most significant baud rate byte
0048 =          RBR     EQU     BASE    ;read buffer register
0048 =          THR     EQU     BASE    ;transmit buffer register
                ;
0100                    ORG     100H
                ;
0100 F3         START:  DI              ;disable interrupts
0101 318303             LXI     SP,STACK
0104 118401             LXI     D,TABLE ;de points to old table
0107 218001             LXI     H,TABLE AND 0FFE0H  ;table add. even 32 bytes
010A E5                 PUSH    H       ;new table address onto stack
010B 0E20               MVI     C,32    ;32 byte count into reg c
010D 1A         SLOOP:  LDAX    D       ;beginning of old table into reg a
010E 77                 MOV     M,A     ;move byte of old table to new table
010F 23                 INX     H       ;update pointer into new table
0110 13                 INX     D       ;update pointer into old table
0111 0D                 DCR     C       ;update 32 byte counter
0112 C20D01             JNZ     SLOOP   ;continue until all 32 bytes moved
0115 E1                 POP     H       ;hl points to beginning of new table
0116 AF                 XRA     A       ;zero reg a
```

```
0117 D34F               OUT     GRPSEL  ;select group 0
0119 7D                 MOV     A,L     ;low address of table in reg a
011A F61E               ORI     D4+LTIM+ADDI+SNGL       ;icw1 set up
011C D34C               OUT     PIC0    ;icw1 out to pic
011E 7C                 MOV     A,H     ;high address of table into hl
011F D34D               OUT     PIC1    ;icw2 out to pic
                ;
0121 3EFF               MVI     A,0FFH  ;mask out all interrupts
0123 D34D               OUT     PIC1    ;send ocw1 to pic-- mask out all
                ;
0125 AF         CINIT:  XRA     A       ;zero reg a
0126 D34F               OUT     GRPSEL  ;select group zero
0128 3E14               MVI     A,TP256 ;256 hz tp signal
012A D34A               OUT     CLK     ;set clock for tp mode
012C 0E20               MVI     C,CSTB  ;clock strobe bit set in reg c
012E A9                 XRA     C       ;put strobe bit high in reg c
012F D34A               OUT     CLK     ;strobe in tp command with strobe high
0131 A9                 XRA     C       ;strobe bbit low again
0132 D34A               OUT     CLK     ;complete setting of tp to 256 hz
                ;
0134 3E01               MVI     A,1     ;group 1 code
0136 D34F               OUT     GRPSEL  ;select uart0
0138 3E87               MVI     A,DLAB+WLS0+WLS1+STB    ;baud rate set up
013A D34B               OUT     LCR     ;set up line control reg for baud rate
013C 3E0C               MVI     A,BRATE AND 0FFH        ;low baud into reg a
013E D348               OUT     DLL     ;baud rate into low baud rate divisor
0140 AF                 XRA     A       ;zero into a
0141 D349               OUT     DLM     ;high baud rate divisor
0143 3E07               MVI     A,WLS0+WLS1+STB ;init. line cntrl. reg for data
0145 D34B               OUT     LCR     ;line control register initialized
0147 3E01               MVI     A,ERBFI ;dav interrupt enabled
  49 D349               OUT     IER     ;output interrupt enable mask
014B DB48               IN      RBR     ;clear dav bit
                ;
014D AF                 XRA     A
014E D34F               OUT     GRPSEL  ;select group 0 for pic command
0150 3E40               MVI     A,IMASK ;interrupt mask for pic
0152 D34D               OUT     PIC1    ;output mask to pic
0154 FB                 EI              ;watch out- interrupts enabled
                        ;
                ;
0155 F3         MAIN:   DI      ;disable interrupts
0156 0E02               MVI     C,2     ;print code for bdos
0158 1E2A               MVI     E,'*'
015A CD0500             CALL    5
015D FB                 EI
015E C35501             JMP     MAIN    ;simple loop
                ;
0161 =          UART1   EQU     $
0161 =          UART2   EQU     $
0161 =          DAISY   EQU     $
                ;
0161 C30000     WRONGO: JMP     0       ;pic picked got interrupt-- warm boot
                ;
0164 0000000000SPACE:   DW      0,0,0,0 ;8 bytes
016C 0000000000         DW      0,0,0,0 ;8 more bytes for a grand total of 16
```

```
0174 0000000000          DW      0,0,0,0 ;8 more bytes for a grand total of 24
017C 0000000000          DW      0,0,0,0 ;8 more bytes for a grand total of 32
                ;
0184 C3A401     TABLE:   JMP     INT0    ;irq0 vector
0187 00                  DB      0       ;1 byte fill
0188 C3B101              JMP     INT1    ;irq1 vector
018B 00                  DB      0       ;1 byte fill
018C C3BE01              JMP     INT2    ;irq2 vector
018F 00                  DB      0       ;1 byte fill
0190 C3E801              JMP     UART0   ;irq3 vector
0193 00                  DB      0       ;1 byte fill
0194 C36101              JMP     UART1   ;irq4 vector
0197 00                  DB      0       ;1 byte fill
0198 C36101              JMP     UART2   ;irq5 vector
019B 00                  DB      0       ;1 byte fill
019C C36101              JMP     DAISY   ;irq6 vector
019F 00                  DB      0       ;1 byte fill
01A0 C3CB01              JMP     CLOCK   ;irq7 vector
01A3 00                  DB      0       ;1 byte fill
                ;
01A4 E5         INT0:    PUSH    H
01A5 D5                  PUSH    D
01A6 C5                  PUSH    B
01A7 F5                  PUSH    PSW
01A8 113702              LXI     D,V0MSG
01AB CD1C02              CALL    PMSG
01AE C30F02              JMP     INTRET  ;return through uart0 mechanism
                ;
01B1 E5         INT1:    PUSH    H
01B2 D5                  PUSH    D
01B3 C5                  PUSH    B
01B4 F5                  PUSH    PSW
01B5 115002              LXI     D,V1MSG
01B8 CD1C02              CALL    PMSG
01BB C30F02              JMP     INTRET  ;return through uart0 mechanism
                ;
01BE E5         INT2:    PUSH    H
01BF D5                  PUSH    D
01C0 C5                  PUSH    B
01C1 F5                  PUSH    PSW
01C2 116902              LXI     D,V2MSG
01C5 CD1C02              CALL    PMSG
01C8 C30F02              JMP     INTRET  ;return through uart0 mechanism
                ;
01CB E5         CLOCK:   PUSH    H
01CC D5                  PUSH    D
01CD C5                  PUSH    B
01CE F5                  PUSH    PSW
01CF AF                  XRA     A
01D0 D34F                OUT     GRPSEL  ;select group0
01D2 3A8202              LDA     TIMER
01D5 3C                  INR     A
01D6 328202              STA     TIMER
01D9 CCE101              CZ      SECONDS
01DC DB4B                IN      CLRCLK  ;remove tp interrupt
01DE C30F02              JMP     INTRET  ;return through uart0 mechanism
```

```
                ;
01E1 0E02       SECONDS:        MVI     C,2
01E3 1E21               MVI     E,'!'
01E5 C30500             JMP     5       ;jump to bdos
                ;
 E8 E5          UART0:  PUSH    H
 E9 D5                  PUSH    D
01EA C5                 PUSH    B
01EB F5                 PUSH    PSW
01EC 3E01               MVI     A,1     ;set up uart0 group
01EE D34F               OUT     GRPSEL  ;select uart0
01F0 DB4A               IN      IIR     ;read interrupt id reg
01F2 F5                 PUSH    PSW     ;save flags
01F3 E601               ANI     INTPEND ;check for valid interrupt
01F5 C20102             JNZ     BADINT  ;this clears dav flag
01F8 F1                 POP     PSW     ;restore flags
01F9 E606               ANI     INTTYP  ;only interested in bits 1 and 2
01FB FE04               CPI     4
01FD CA0802             JZ      NEWCH
0200 F5                 PUSH    PSW     ;if not dav then bad interrupt
                ;
0201 F1         BADINT: POP     PSW
0202 112102             LXI     D,BAD
0205 CD1C02             CALL    PMSG
                ;
0208 DB48       NEWCH:  IN      RBR     ;read character from selected uart
020A D348               OUT     THR     ;echo character from selected uart
020C C30F02             JMP     INTRET  ;exit through interrupt return routine
                ;
                ;routine below is a good general purpose exit routine
                ;
 0F AF          INTRET: XRA     A       ;select group0
0210 D34F               OUT     GRPSEL
0212 3E20               MVI     A,EOI   ;signal end of interrupt to pic
0214 D34C               OUT     PIC0
0216 F1                 POP     PSW
0217 C1                 POP     B
0218 D1                 POP     D
0219 E1                 POP     H
021A FB                 EI              ;enable interrupts
021B C9                 RET             ;back to main program or next interrupt
                ;
021C 0E09       PMSG:   MVI     C,9     ;print string pointed to by de
021E C30500             JMP     5       ;and ending with $
                ;
0221 0D0A       BAD:    DB      ACR,ALF
0223 496C6C6567         DB      'Illegal interrupt'
0234 0D0A24             DB      ACR,ALF,'$'
                ;
0237 0D0A       V0MSG:  DB      ACR,ALF
0239 766563746F         DB      'vectored interrupt 0'
024D 0D0A24             DB      ACR,ALF,'$'
                ;
0250 0D0A       V1MSG:  DB      ACR,ALF
0252 766563746F         DB      'vectored interrupt 1'
0266 0D0A24             DB      ACR,ALF,'$'
```

```
                        ;
0269 0D0A               V2MSG:  DB      ACR,ALF
026B 766563746F                 DB      'vectored interrupt 2'
027F 0D0A24                     DB      ACR,ALF,'$'
                        ;
0282 00                 TIMER:  DB      0       ;initial timer value
                        ;
0283                            DS      100H    ;room for stack
0383 =                  STACK   EQU     $       ;top of stack goes here

A>
```

Software Samples

The following program both sets and reads the clock/calender of the MULT/IO board. The program runs under CP/M and assumes the MULT/IO board to be adressed at I/O port 48h.

To set the time using this program, type:

WATCH www MMM dd hh mm ss (pm/am)

where 'www' are the first three letters of the day of the week, 'MMM' are the first three letters of the month, 'dd' are the decimal day of the month, 'hh' are the decimal hour of the day, 'mm' are the decimal minutes of the hour, and 'ss' are the decimal seconds of the minute. Twelve hour format may be used if either 'PM' or 'AM' is typed at the end of this string, otherwise data will be assumed to be in 24 hour format. Spaces should separate the data fields. Day of week and month of year may exceed three characters, but only the first three will be analyzed. Leading zero's may be omitted as long as one character appears in the field in question.

For example, typing:

WATCH MON NOV 17 7 30 0 AM

would set the clock/calender to Monday, November 17, 7:30:00 a.m.

To read the clock, simply type:

WATCH

```
*****************************************************************
*                                                               *
* Time display/set program for Thinker Toys Mult/IO board.      *
*                                                               *
* Bobby Dale Gifford.                                           *
* 9/25/80                                                       *
*                                                               *
*****************************************************************

rev     equ     10              ;Revision # x.x

base    equ     48h             ;Base of Mult I/O ports
grpsel  equ     base+7          ;Group select
clk     equ     base+2          ;Clock port
clkclk  equ     2               ;Clock clk bit
clkcl   equ     8               ;Clock cl bit
rclk    equ     0ch             ;Read clock command
cstb    equ     20h             ;Clock strobe bit
shft    equ     4               ;Shift bits command
tp64    equ     10h             ;Output tick pulse at 64 hz
reghld  equ     0               ;Register hold command
wclk    equ     8               ;Write clock command

bdos    equ     5               ;Bdos entry point
cbuff   equ     81h             ;Command buffer string
clen    equ     80h             ;Command length byte
wboot   equ     0               ;Warm boot location
const   equ     11              ;Get constat function #
pstr    equ     9               ;Print string function #
readcon equ     10              ;Read console buffer
acr     equ     0dh             ;Carriage return
alf     equ     0ah             ;Line feed

        org     100h            ;Transient program area

start   lhld    bdos+1          ;Set up stack
        sphl
        call    skipb           ;Skip command line blanks
        jz      display         ;No command line

sett    lxi     h,days          ;Array of string pointers to match
        call    match3          ;Look for match
        jz      exit            ;No match
        lxi     d,-days         ;Form index
        dad     d
        mov     a,l             ;Get low byte
        stc                     ;Clear the carry
        cmc
        rar                     ;Divide index by 2
        sta     mthday          ;Day of week finished

        lxi     h,months        ;Array of string pointers to match
        call    match3          ;Look for match
        jz      exit            ;No match
        lxi     d,-months       ;Form index
```

```
        dad     d
        mov     a,l             ;Get low byte
        stc                     ;Clear the carry
        cmc
        ral
        ral
        ral
        mov     b,a             ;Save in B
        lda     mthday          ;Or in with day
        ora     b
        sta     mthday
        call    bcd2            ;Scan for two valid bcd digits
        jc      exit
        sta     date            ;New date
        call    bcd2            ;Scan for two more valid bcd digits
        jc      exit
        sta     hour            ;New hour
        call    bcd2            ;Scan for two more valid bcd digits
        jc      exit
        sta     minutes         ;New minutes
        call    bcd2            ;Scan for last valid bcd digits
        jc      exit
        sta     seconds         ;New seconds
        call    skipb           ;Skip trailing blanks
        jz      noap
        call    scan
        cpi     'P'             ;Check for AM or PM
        push    psw
        cz      uphrs
        pop     psw
        cpi     'A'
        cz      dwnhrs
        call    skipc
        call    skipb
        jnz     exit            ;If anything remaining, then error

noap    mvi     a,reghld        ;Issue register hold command
        call    setup
        mvi     a,tp64          ;Set up clock pulse
        call    setup
        lxi     d,waitmsg       ;Wait for carriage return
        call    pmsg
        lxi     d,ibuff         ;Read console
        mvi     c,readcon
        call    bdos
        call    writec          ;Write the time
        lxi     d,acralf
        call    pmsg
        call    displl          ;Display the current time
        jmp     wboot           ;All done

;*****************************************************************
;*                                                               *
;* Writec does the actual clock time writing. This routine must  *
;* not be interrupted.                                           *
```

```
*                                                                    *
**********************************************************************

writec  xra     a               ;Select group 0
        out     grpsel
        mvi     a,shft          ;Shift command
        call    setup
        push    h               ;Save clock data address
wbyte   mvi     e,8             ;Bit shift counter
        inx     h               ;Bump to next byte of data
wbit    mov     a,m             ;Get current byte of data
        rar                     ;LSB into carry
        mov     m,a             ;Save current byte
        ral                     ;Carry into LSB
        ani     1               ;Through away useless bits
        xthl                    ;Recover address of clock data
        ora     m               ;Get current state
        xthl                    ;Recover current byte counter
        call    clkstb          ;Strobe in one bit
        dcr     e               ;Update bit counter
        jnz     wbit            ;Same byte ?
        dcr     d               ;Update bye counter
        jnz     wbyte           ;All done ?
        pop     h               ;Recover address of clock data
        mov     a,m             ;Get current state
        ori     wclk            ;Set write clock bit
        call    clkcmd          ;Issue write time command
        xri     wclk            ;Turn off write time command
        jmp     clkcmd

**********************************************************************
*                                                                    *
* Bcd2 scans the command line for up to two valid ascii digits       *
* and returns the result as a packed bcd byte in reg A.              *
*                                                                    *
**********************************************************************

bcd2    call    skipb           ;Skip any preceeding blanks
        call    scan            ;Get first char of day of month
        stc                     ;Carry is error
        rz
        cpi     ':'
        jz      bcd2
        cpi     ','
        jz      bcd2
        call    digit           ;Check for valid decimal digit
        rc
        mov     b,a             ;Save in B
        call    scan
        jz      okd
        cpi     ','             ;Check for end of day of month
        jz      okd
        cpi     ' '
        jz      okd
        cpi     ':'
```

```
        jz      okd
        call    digit
        rc
        stc                     ;Clear the carry
        cmc
        push    psw             ;Save low nibble
        mov     a,b
        ral                     ;Put previous digit into high nibble
        ral
        ral
        ral
        mov     b,a             ;Save in B
        pop     psw             ;Recover low digit
        ora     b               ;Form byte
        mov     b,a             ;Save in B
okd     mov     a,b             ;Recover day of month
        stc                     ;No error
        cmc
        ret

;*****************************************************************
;*                                                               *
;* Digit checks if the char in reg A is a valid ascii digit.     *
;*                                                               *
;*****************************************************************

digit   cpi     '0'             ;Less than 0
        rc
        cpi     '9'+1           ;Greater than 9
        cmc
        rc
        sui     '0'             ;Strip off ascii bias
        ret

;*****************************************************************
;*                                                               *
;* Match3 guarantees that at least three characters are matched  *
;* with the command line.                                        *
;*                                                               *
;*****************************************************************

match3  mvi     a,3             ;Clear match count
        sta     mcnt
        mov     e,m             ;Get current string pointer
        inx     h
        mov     d,m
        inx     h
        mov     a,e             ;Check if all done
        ora     d
        rz                      ;No match
        push    h               ;Save current array pointer
        lhld    scanpnt         ;Save current scan pointer
        push    h
        lda     clen            ;Save current command length
        push    psw
```

```
mtchmo  call    scan            ;Scan and convert to upper case
        jz      nomatch         ;No match if out of chars
        call    toupper
        mov     b,a             ;Save in B
        ldax    d               ;Get next char in string
        inx     d               ;Bump string pointer
        call    toupper         ;Convert to upper case
        cmp     b               ;Does it match ?
        jnz     nomatch         ;No match
        lda     mcnt            ;Get match count
        dcr     a               ;Matched three ?
        sta     mcnt            ;Save match count
        jnz     mtchmo          ;Match more ?
        call    skipc           ;Skip rest of characters
        pop     h               ;Throw away old scan pointer
        pop     h               ;Throw away old command length
        pop     h               ;Recover array pointer
        dcx     h               ;Backup array pointer
        dcx     h
        rnz                     ;No error return
        inr     a               ;No error return
        ret
nomatch pop     psw             ;Recover command length
        sta     clen            ;Restore command length
        pop     h               ;Recover scan pointer
        shld    scanpnt         ;Restore scan pointer
        pop     h               ;Recover array pointer
        jmp     match3          ;Try again

*****************************************************************
*                                                               *
* Display continually displays the time as long as nothing is   *
* typed on the console.                                         *
*                                                               *
*****************************************************************

display call    displl          ;Display one time line
        mvi     c,const         ;Check console for char
        call    bdos
        ana     a               ;If anything typed then reboot
        jnz     wboot
        lxi     d,acrmsg        ;Print carriage return only
        call    pmsg
        jmp     display         ;Go print the time again

*****************************************************************
*                                                               *
* Displl displays the current time once.                        *
*                                                               *
*****************************************************************

displl  call    readc           ;Read the clock - watch out if interrupts or
        lda     mthday          ;Get the day of the week
        ani     7               ;Through away irrelevent bits
okday   ral                     ;Multiply by 2
```

```
mov     e,a             ;Form 16 bit offset
mvi     d,0
lxi     h,days          ;Array of string pointers
dad     d               ;Form absolute address of string
mov     e,m             ;Get low string address byte
inx     h               ;Point to high byte
mov     d,m             ;Get high byte
mov     a,e             ;Check for invalid day
ora     d
jz      displ1          ;Start over again if invalid
call    pmsg            ;Print the day

lda     mthday          ;Get the month
rar                     ;Adjust for proper offset
rar
rar
ani     1eh             ;Multiply by two and throw out
                        ;       irrelevent bits
mov     e,a             ;Form 16 bit offset
mvi     d,0
lxi     h,months        ;Array of string pointers
dad     d               ;Form absolute address of string
mov     e,m             ;Get low string address byte
inx     h               ;Point to high byte
mov     d,m             ;Get high byte
mov     a,d             ;Check for invalid month
ora     e
jz      displ1          ;Start over again if invalid
call    pmsg            ;Print the month

lxi     h,tbuff         ;Pointer to temporary storage
push    h               ;Save for printing
lda     date            ;Convert the date to ascii
rar                     ;Get high digit into low nibble
rar
rar
rar
ani     0fh
cnz     putlow          ;Don't print leading zero
lda     date            ;Get the low digit
call    putlow          ;Stuff it in the buffer
mvi     a,','           ;And the comma and space
call    put
mvi     a,' '
call    put

lda     hour            ;Get the hour
cpi     13h             ;Check for AM or PM
cnc     subhr           ;Convert PM from 13-24 into 0-12
ora     a               ;Check for 12 midnight
cz      mak12
call    puthi           ;Put both digits into the buffer
mvi     a,':'           ;Put the colon in the buffer
call    put
lda     minutes         ;Get the minutes
```

```
        call    puthi           ;Put both minutes digits in the buffer
        mvi     a,':'           ;Put another colon in the buffer
        call    put
        lda     seconds         ;Get the seconds
        call    puthi           ;Put both second digits in the buffer
        mvi     a,' '           ;One space into the buffer
        call    put
        lda     hour            ;Check hours for AM or PM
        cpi     12h
        mvi     a,'a'           ;Print 'A' or 'P'
        jc      isam
        mvi     a,'p'
isam    call    put             ;Put the 'A' or 'P' in the buffer
        mvi     a,'m'           ;Put the 'M' in the buffer
        call    put
sploop  mov     a,m             ;Get the next char in the buffer
        cpi     '$'             ;Is it the end ?
        jz      endsp           ;All done
        mvi     a,' '           ;Get a space
        call    put             ;Put it in the buffer
        jmp     sploop          ;Finish padding with spaces

endsp   pop     d               ;Recover the Buffer address
        jmp     pmsg            ;Print the buffer

*************************************************************************
*                                                                       *
* Readc does the actual clock reading (40 bits) from the                *
* hardware. If interrupts are enabled, then care must be taken          *
* to assure that this routine is not interrupted until it               *
* completes.                                                            *
*                                                                       *
*************************************************************************

readc   xra     a               ;Select group zero
        out     grpsel.
        mvi     a,rclk          ;Read clock into 40 bit shift register
        call    setup
        push    h               ;Save address of clkdata
        xri     clkcl           ;Issue shift command
        call    clkcmd
rbyte   mvi     e,8             ;Prep for 8 bits
        inx     h               ;Bump to next address of clock data
rbit    in      clk             ;Read one bit
        rar                     ;Put bit into carry
        mov     a,m             ;Get partially assembled byte
        rar                     ;Shift in the bit just read
        mov     m,a             ;Save partially assembled byte
        xthl                    ;Get address of clkdata
        mov     a,m             ;Get clock data
        xthl                    ;Save address of clock data
        call    clkstb          ;Strobe the shift register
        dcr     e               ;All done with this byte ?
        jnz     rbit            ;Read another bit if not
        dcr     d               ;Completely done ?
```

```
        jnz     rbyte           ;Read another byte if not
        pop     h               ;Recover address of clkdata
clkcmd  mvi     c,cstb          ;Get clock strobe bit
clkstb  out     clk             ;Output strobe low
        call    delay           ;Wait for chip to see the strobe low
        xra     c               ;Turn strobe high
        out     clk             ;Output strobe high
        call    delay           ;Wait for chip to see the strobe high
        xra     c               ;Turn strobe low
        out     clk             ;Output strobe low
        call    delay
        mvi     c,clkclk        ;Clock clk bit
        ret

setup   mvi     d,5             ;Count of bytes to read
        lxi     h,clkdata       ;Address of clock data
        ora     m               ;Get current bit state
        jmp     clkcmd          ;Issue the command

delay   mvi     b,0             ;Worst case is 700 usec
delay1  dcr     b
        jnz     delay1
        ret

*****************************************************************
*                                                               *
* Puthi puts the high and low nibbles of the bcd number in      *
* the a reg in the temporary buffer.                            *
*                                                               *
*****************************************************************

puthi   push    psw             ;Save low nibble
        rar                     ;Put high nibble into low nibble
        rar
        rar
        rar
        call    putlow          ;Print the low nibble of a reg
        pop     psw             ;Recover the low nibble
putlow  ani     0fh             ;Strip off irrelevent bits
        adi     '0'             ;Form Ascii character
put     mov     m,a             ;Put char in buffer
        inx     h               ;Bump buffer pointer
        ret

*****************************************************************
*                                                               *
* Exit is the standard error message for invalid command.       *
*                                                               *
*****************************************************************

exit    lxi     d,badtmsg
        call    pmsg
        jmp     wboot

*****************************************************************
```

```
*                                                                     *
* Pmsg is the CP/M print string function.                             *
*                                                                     *
***********************************************************************

pmsg    mvi     c,pstr
        jmp     bdos

subhr   adi     88h             ;Subhr adjusts the BCD number to
        daa                     ;       be between 1 and 12
        ret

makl2   mvi     a,12h
        ret

uphrs   lda     hour
        cpi     12h
        rz
        adi     12h
        sta     hour
        ret

dwnhrs  lda     hour
        cpi     12h
        rnz
        xra     a
        sta     hour
        ret

skipc   call    scan            ;Get next char
        rz                      ;Return if no more chars
        cpi     ' '             ;Check for space
        jnz     skipc           ;Continue if not
        ret

skipb   call    scan            ;Get next char
        rz                      ;Return if no characters left
        cpi     ' '             ;Is it a space
        jz      skipb           ;Skip it
unscan  push    h               ;Save HL
        lhld    scanpnt         ;Get command scan pointer
        dcx     h               ;Back it up
        shld    scanpnt         ;Save updated char
        lda     clen            ;Update length
        inr     a
        sta     clen            ;Save updated length
        pop     h               ;Restore HL
        ret

scan    lda     clen            ;Check if anything left
        ana     a
        rz                      ;Return with Z set if no more
        dcr     a               ;Update length
        sta     clen
        push    h               ;Save HL
```

```
        lhld    scanpnt         ;Get command pointer
        mov     a,m
        inx     h               ;Update command pointer
        shld    scanpnt
        pop     h
        ora     a               ;Clear Z flag
        ret

toupper cpi     'a'             ;Is it lower case ?
        rc
        cpi     'z'+1
        rnc
        sui     ' '
        ret

;*****************************************************************
;*                                                               *
;* The following are data used within the program.               *
;*                                                               *
;*****************************************************************

clkdata: db     0               ;Current state of clk port
seconds: db     0               ;Seconds read
minutes: db     0               ;Minutes read
hour    db      0               ;Hours read
date    db      0               ;Date read
mthday  db      0               ;Week day and month read

;*****************************************************************
;*                                                               *
;* Days is an array of pointers to strings, used to print the    *
;* english version of the day of the week.                       *
;*                                                               *
;*****************************************************************

days    dw      sun
        dw      mon
        dw      tue
        dw      wed
        dw      thu
        dw      fri
        dw      sat
        dw      0                       ;Illegal day

sun     db      'Sunday, $'
mon     db      'Monday, $'
tue     db      'Tuesday, $'
wed     db      'Wednesday, $'
thu     db      'Thursday, $'
fri     db      'Friday, $'
sat     db      'Saturday, $'

;*****************************************************************
;*                                                               *
;* Months is an array of pointers to strings, used to print the  *
```

```
* english version of the month of the year.                          *
*                                                                    *
**********************************************************************

months  dw      jan
        dw      feb
        dw      mar
        dw      apr
        dw      may
        dw      jun
        dw      jul
        dw      aug
        dw      sep
        dw      oct
        dw      nov
        dw      dec
        dw      0,0,0,0                 ;Illegal months

jan     db      'January $'
feb     db      'February $'
mar     db      'March $'
apr     db      'April $'
may     db      'May $'
jun     db      'June $'
jul     db      'July $'
aug     db      'August $'
sep     db      'September $'
oct     db      'October $'
nov     db      'November $'
dec     db      'December $'
acrmsg  db      acr,'$'
acralf  db      acr,alf,'$'

**********************************************************************
*                                                                    *
* Tbuff is used to prepare the day of the month, hours, minutes,     *
* and seconds prior to printing.                                     *
*                                                                    *
**********************************************************************

tbuff   db      '00, 00:00:00 am                            $'

badtmsg db      acr,alf
        db      'Invalid Time specified.$'

waitmsg db      acr,alf
        db      'Press return to set the time: $'

ibuff   db      10,10
        ds      10

scanpnt dw      cbuff
mcnt    db      0

        end
```

Software Samples

The following program is an example of the use of the Daisy ports of the MULT/IO board. The program assumes there is a standard Diablo Htype II connected to the 50 pin ribbon cable.

```
                ;*****************************************************************
                ;*                                                               *
                ;* Diablo 1610 simulator for the Morrow Designs / Thinker Toys   *
                ;* Mult I/O board. The simulator makes the parallel Hityp II     *
                ;* look like a serial 1610.                                      *
                ;*                                                               *
                ;* This interface is designed to work with the INSTALL.COM       *
                ;* program which is available from Morrow Designs / Thinker      *
                ;* Toys. For an explanation of how this works consult the        *
                ;* INSTALL documentation.                                        *
                ;*                                                               *
                ;* Bobby Dale Gifford.                                           *
                ;* 10/13/80                                                      *
                ;*                                                               *
                ;*****************************************************************

                ;*****************************************************************
                ;*                                                               *
                ;* Special character equates.                                    *
                ;*                                                               *
                ;*****************************************************************

000D =          ACR     EQU     0DH             ;Carriage Return
000A =          ALF     EQU     0AH             ;Line feed
0003 =          AETX    EQU     3               ;ETX character
0006 =          AACK    EQU     6               ;ACK character
001B =          AESC    EQU     33Q             ;Escape character
0008 =          ABS     EQU     10Q             ;Back Space
0009 =          AHT     EQU     11Q             ;Horizontal tab
000C =          AFF     EQU     14Q             ;Form Feed
0007 =          ABEL    EQU     7               ;Bell
0020 =          ASP     EQU     40Q             ;Space
0000 =          ANUL    EQU     0               ;Null
007F =          ADEL    EQU     177Q            ;Delete
001E =          ARS     EQU     36Q             ;RS character
001F =          AUS     EQU     37Q             ;US character
000B =          AVT     EQU     13Q             ;Vertical tab

                ;*****************************************************************
                ;*                                                               *
                ;* The following equates are for the Mult I/O board.             *
                ;*                                                               *
                ;*****************************************************************

0007 =          GRPSEL  EQU     7               ;Group select port offset
0004 =          PIC0    EQU     4               ;Interupt controller port 0
0005 =          PIC1    EQU     5               ;Interupt controller port 1
0001 =          DAISY1  EQU     1               ;Daisy wheel port 1
0040 =          IMASK   EQU     40H             ;Interupt enable mask

0010 =          D4      EQU     10H
0008 =          LTIM    EQU     8               ;Level trigered interrupt mode
0004 =          ADDI    EQU     4               ;Address interval
0002 =          SNGL    EQU     2               ;Single 8259
0066 =          EOI6    EQU     66H             ;End of interupt 6
```

```
0080 =          RSTBIT  EQU     080H            ;Restore Bit
0004 =          DATA11  EQU     4               ;Data bits on daisy port
0008 =          DATA12  EQU     8
0003 =          DATA910 EQU     3
1020 =          CRSTRD  EQU     1020H           ;Carriage ready
 0 =            PFSTRD  EQU     810H            ;Paper feed ready
2040 =          PWSTRD  EQU     2040H           ;Print wheel ready

                ;*****************************************************************
                ;*                                                               *
                ;* Below is a standard CP/M Cbios jump table as required by      *
                ;* INSTALL.                                                      *
                ;*                                                               *
                ;*****************************************************************

0000 C30000             JMP     $               ;No change in the cold boot
0003 C3F801     OWBOOT: JMP     NWBOOT          ;New warm boot routine
0006 C32F03     OCONST: JMP     NCONST          ;New console status
0009 C33F03     OCONIN: JMP     NCONIN          ;New console input
000C C30C00             JMP     $               ;No change in the console output
000F C35803             JMP     NLIST           ;New list device output
0012 C31200             JMP     $               ;No change in the punch device outpu
0015 C31500             JMP     $               ;No change in the reader device outp
0018 C31800             JMP     $               ;No change in the home routine
001B C31D03     OSEL:   JMP     NSEL            ;New seldsk routine
001E C31E00             JMP     $               ;No change in the settrk
0021 C32100             JMP     $               ;No change in the setsec
0024 C32400             JMP     $               ;No change in the setdma
0027 C32303     ORD:    JMP     NRD             ;New read
002A C32903     OWR:    JMP     NWR             ;New write
 2D C39603              JMP     NLSTST          ;New list device status
 30 C33000              JMP     $               ;No change in the sectran

                ;*****************************************************************
                ;*                                                               *
                ;* The following routines are for handshaking with the printer   *
                ;* they can be used directly or by the CBIOS of CP/M.            *
                ;*                                                               *
                ;*****************************************************************

0033 C31F04     REST:   JMP     RESTOR          ;Initialization procedure
0036 C3E803     LST:    JMP     LSTDEV          ;Printer character output
                                                ;       character in reg C
0039 C3A003     HNDXOF: JMP     XONOFF          ;Printer busy test XON/XOFF
                                                ;       returns with:
                                                ;               A = 1 queue full
                                                ;               A = 0 queue not full
                                                ;               A = 0ffh queue empty
003C C3B803     HNDETX: JMP     ETXACK          ;ETX and ACK software handshake
                                                ;       returns with:
                                                ;               A = 0 No ACK to tran
                                                ;               A = 0ffh ACK transmi

                ;*****************************************************************
                ;*                                                               *
                ;* Dynamic data locations used by the simulator.                 *
```

```
                        ;*                                                    *
                        ;*****************************************************
000A =          CPERI   EQU     10              ;Default to 10 characters per inch
0006 =          LPERI   EQU     6               ;Default lines per inch
0078 =          HINC    EQU     120             ;Horizontal increments per inch
0030 =          VINC    EQU     48              ;Vertical increments per inch
00A0 =          NUMTABS EQU     160             ;Number of horizontal tabs
0096 =          MAXCHRS EQU     150             ;Maximum number of printer character
0630 =          MAXRGT  EQU     1584            ;Maximum carriage position

003F 02         ACKXON: DB      2               ;Default handshake is XON/XOFF
                                                ;       Can be changed with -Hx.
                                                ;       Possible handshakes are:
                                                ;               0 = none
                                                ;               1 = ETX/ACK
                                                ;               2 = XON/XOFF
                                                ;               3 = ETX/ACK through
                                                ;                   (made for electr
0040 48         BASE:   DB      48H             ;Default Mult I/O board base address
                                                ;       Can be changed with -bxx.
0041 6E00       DFRMLN: DW      110             ;Default forms length 10 times the f
                                                ;       length switch. Can be change
                                                ;       with -fxx.
0043 0A00       DSPACE: DW      CPERI           ;Default characters per inch.
                                                ;       Can be changed with -cxx.
0045 0600       DLINES: DW      LPERI           ;Default lines per inch.
                                                ;       Can be changed with -lxx.
0047 00         AUTOLF: DB      0               ;Default to no Auto line feed.
                                                ;       Can be changed with -ax.
0048 0000       HMI:    DW      0               ;Horizontal motion index. Set by ^ S
                                                ;       and escape sequences.
004A 0000       VMI:    DW      0               ;Vertical motion index. Set by RESTO
                                                ;       and escape sequences.
004C 0000       VPOS:   DW      0               ;Vertical position. Set by platen mo
004E 0000       DLVPOS: DW      0               ;Delta vpos. Set by platen motion
0050 0000       HPOS:   DW      0               ;Horizontal position. Set by carriag
0052 0000       DLHPOS: DW      0               ;Delta hpos. Set by carriage motion
0054 0000       LMAR:   DW      0               ;Left margin
0056 00         DIRFLG: DB      0               ;Direction flag
0057 00         GRHFLG: DB      0               ;Graphics mode flag
0058 00         ESCFLG: DB      0               ;Escape sequence in progress flag
0059 00         ETXFLG: DB      0               ;Used for ETX/ACK handshake
005A 00         HNDFLG: DB      0               ;Handshake in progress flag
005B            TABSTP: DS      NUMTABS         ;Tab stops array

00FB            QUEUE:  DS      MAXCHRS         ;Circular Queue of printer character
0191 FB00       QUETOP: DW      QUEUE           ;Queue top pointer
0193 FB00       QUEBOT: DW      QUEUE           ;Queue bottom pointer

                        ;*
                        ;* The following data only needs to be included if the 8259
                        ;* has not been initialized.
                        ;*
0195 0000000000         DW      0,0,0,0
019D 0000000000         DW      0,0,0,0
```

```
01A5 0000000000          DW      0,0,0,0
01AD 0000000000          DW      0,0,0,0
01B5 C3E904     TABLE:   JMP     NOINT          ;No interupt
01B8 00                  DB      0
^1B9 C3E904              JMP     NOINT
 .BC 00                  DB      0
01BD C3E904              JMP     NOINT
01C0 00                  DB      0
01C1 C3E904              JMP     NOINT
01C4 00                  DB      0
01C5 C3E904              JMP     NOINT
01C8 00                  DB      0
01C9 C3E904              JMP     NOINT
01CC 00                  DB      0
01CD C3EC04              JMP     PWINT
01D0 00                  DB      0
01D1 C3E904              JMP     NOINT
01D4 00                  DB      0

01D5 0000       HLSAVE:  DW      0              ;Used by interupt routine
01D7 00         AFSAVE:  DB      0              ;Used by interupt routine
01D8 8000       SCSTUF   DW      80H            ;Scan buffer data

01DA                     DS      30             ;Stack space
01F8 =          STACK    EQU     $

                ;*****************************************************************
                ;*                                                               *
                ;* New Boot routine, examine the command line put at 80H by      *
                ;* install.                                                      *
                ;*                                                               *
                ;*****************************************************************

01F8 3E01       NWBOOT:  MVI     A,1            ;Is this a second warm boot ?
01F9 =          WBFLG    EQU     $-1
01FA A7                  ANA     A
01FB 3E00                MVI     A,0            ;Reset the warm boot flag
01FD 32F901              STA     WBFLG
0200 CA0300              JZ      OWBOOT         ;Don't reset if second warm boot
0203 C31102              JMP     SKPDSH

0206 CD3E02     CLOOP:   CALL    SCAN
0209 CA3802              JZ      NOMORE
020C FE2D                CPI     '-'            ;Check for flag
020E C20602              JNZ     CLOOP
0211 CD3E02     SKPDSH:  CALL    SCAN
0214 CA3802              JZ      NOMORE
0217 FE48                CPI     'H'            ;New handshake routine -Hx
0219 CC4D02              CZ      NEWH
021C FE42                CPI     'B'            ;New I/O base -Bxx
021E CC6302              CZ      NEWB
0221 FE46                CPI     'F'            ;New forms length -Fxx
0223 CC8A02              CZ      NEWF
0226 FE43                CPI     'C'            ;New characters per inch -Cxx
0228 CCC102              CZ      NEWC
022B FE4C                CPI     'L'            ;New lines per inch -Lxx
```

```
022D CCE402             CZ      NEWL
0230 FE41               CPI     'A'             ;New auto line feed -Ax
0232 CCF002             CZ      NEWA
0235 C30602             JMP     CLOOP


0238 CD3300     NOMORE: CALL    REST            ;Reset the printer
023B C30300             JMP     OWBOOT          ;Go to the warm boot

023E E5         SCAN:   PUSH    H               ;Return the next character in the c
023F 2AD801             LHLD    SCSTUF          ;Pointer to next char
0242 7E                 MOV     A,M             ;Get next char
0243 A7                 ANA     A               ;Test error return
0244 CA4B02             JZ      NOUPDT          ;No update
0247 23                 INX     H               ;Update pointer
0248 22D801             SHLD    SCSTUF          ;Save new pointer
024B E1         NOUPDT: POP     H               ;Restore registers
024C C9                 RET

024D CD3E02     NEWH:   CALL    SCAN            ;End of command ?
0250 C8                 RZ
0251 FE31               CPI     '1'
0253 DA6002             JC      ZRET            ;Invalid ?
0256 FE34               CPI     '3'+1
0258 D26002             JNC     ZRET
025B D630               SUI     '0'
025D 323F00             STA     ACKXON          ;Set new handshake option
0260 3E00       ZRET:   MVI     A,0
0262 C9                 RET

0263 CD3E02     NEWB:   CALL    SCAN            ;End of command ?
0266 C8                 RZ
0267 CD0503             CALL    OKHEX           ;Valid hex character ?
026A DA6002             JC      ZRET
026D 17                 RAL
026E 17                 RAL
026F 17                 RAL
0270 17                 RAL
0271 47                 MOV     B,A
0272 CD3E02             CALL    SCAN
0275 C8                 RZ
0276 CD0503             CALL    OKHEX           ;Valid hex character ?
0279 DA6002             JC      ZRET
027C B0                 ORA     B
027D 47                 MOV     B,A
027E E607               ANI     7               ;Check if divisible by 8
0280 C26002             JNZ     ZRET
0283 78                 MOV     A,B
0284 324000             STA     BASE            ;New I/O base
0287 C36002             JMP     ZRET

028A CD9C02     NEWF:   CALL    GETTWO          ;New default forms length
028D DA6002             JC      ZRET
0290 110A00             LXI     D,10            ;Set to ten times the forms length
0293 CDEA07             CALL    HLTDE
0296 224100             SHLD    DFRMLN
```

```
0299 C36002             JMP     ZRET

029C CD3E02     GETTWO: CALL    SCAN            ;Get two decimal digits
029F CABF02             JZ      NOGD            ;No digits
02A2 CD1303             CALL    OK09            ;Check for 0-9
02A5 DABF02             JC      NOGD
02A8 87                 ADD     A               ;Multiply by 10
02A9 47                 MOV     B,A
02AA 87                 ADD     A
02AB 87                 ADD     A
02AC 80                 ADD     B
02AD 47                 MOV     B,A
02AE CD3E02             CALL    SCAN            ;Get next character
02B1 CABF02             JZ      NOGD            ;No character
02B4 CD1303             CALL    OK09            ;Check if 0-9
02B7 DABF02             JC      NOGD            ;No good
02BA 80                 ADD     B               ;Add into result
02BB 6F                 MOV     L,A
02BC 2600               MVI     H,0             ;Make it a 16 bit number
02BE C9                 RET
02BF 37         NOGD:   STC                     ;Error return
02C0 C9                 RET

02C1 CD3E02     NEWC:   CALL    SCAN            ;Change the default characters per
02C4 C8                 RZ
02C5 FE31               CPI     '1'             ;Must be 10 or 12
02C7 C26002             JNZ     ZRET
02CA CD3E02             CALL    SCAN
02CD C8                 RZ                      ;Only one character
02CE FE30               CPI     '0'
02D0 2E0A               MVI     L,10            ;It was ten
02D2 CADC02             JZ      NEWCOK
02D5 FE32               CPI     '2'
02D7 2E0C               MVI     L,12            ;It was 12
02D9 C26002             JNZ     ZRET
02DC 2600       NEWCOK: MVI     H,0             ;Make 16 bit integer
02DE 224300             SHLD    DSPACE
02E1 C36002             JMP     ZRET

02E4 CD9C02     NEWL:   CALL    GETTWO          ;New lines per inch
02E7 DA6002             JC      ZRET            ;Error reading digits
02EA 224500             SHLD    DLINES
02ED C36002             JMP     ZRET

02F0 CD3E02     NEWA:   CALL    SCAN            ;New auto line feed
02F3 FE31               CPI     '1'             ;Must be 0 or 1
02F5 CAFD02             JZ      NEWAOK
02F8 FE30               CPI     '0'
02FA C26002             JNZ     ZRET
02FD D630       NEWAOK: SUI     '0'             ;Set the auto flag
02FF 324700             STA     AUTOLF
0302 C36002             JMP     ZRET

0305 CD1303     OKHEX:  CALL    OK09            ;Check first if 0-9
0308 D0                 RNC                     ;Yes
0309 FE41       OKAF:   CPI     'A'             ;Check if less than 'A'
```

```
030B D8                 RC
030C FE47               CPI     'F'+1           ;Check if greater than 'F'
030E 3F                 CMC
030F D8                 RC
0310 D64B               SUI     'A'+10          ;Make into binary
0312 C9                 RET

0313 FE30       OK09:   CPI     '0'             ;Check for 0-9
0315 D8                 RC                      ;Less than '0'
0316 FE3A               CPI     '9'+1           ;Check if greater than '9'
0318 3F                 CMC
0319 D8                 RC
031A D630               SUI     '0'             ;Turn into binary
031C C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* New select disk routine, disable interupts.                   *
                ;*                                                               *
                ;*****************************************************************

031D F3         NSEL:   DI
031E CD1B00             CALL    OSEL            ;Execute old disk select
0321 FB                 EI
0322 C9                 RET

0323 F3         NRD:    DI                      ;Execute old disk read
0324 CD2700             CALL    ORD
0327 FB                 EI
0328 C9                 RET

0329 F3         NWR:    DI
032A CD2A00             CALL    OWR             ;Execute old disk write
032D FB                 EI
032E C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* New console status routine, used with ETX/ACK handshake.      *
                ;*                                                               *
                ;*****************************************************************

032F 3A3F00     NCONST: LDA     ACKXON
0332 FE03               CPI     3
0334 C20600             JNZ     OCONST
0337 CD0600             CALL    OCONST          ;Check old console status
033A A7                 ANA     A
033B C0                 RNZ
033C C33C00             JMP     HNDETX          ;Check ETX handshake

                ;*****************************************************************
                ;*                                                               *
                ;* New console input routine, used with ETX/ACK handshake.       *
                ;*                                                               *
                ;*****************************************************************
```

```
033F 3A3F00    NCONIN: LDA     ACKXON          ;Determine the type of handshake
0342 FE03              CPI     3
0344 C20900            JNZ     OCONIN          ;None, do old conin
0347 CD0600            CALL    OCONST
034A A7                ANA     A
034B C20900            JNZ     OCONIN
034E CD3C00            CALL    HNDETX
0351 A7                ANA     A
0352 3E06              MVI     A,AACK
0354 CA3F03            JZ      NCONIN
0357 C9                RET

                ;*****************************************************************
                ;*                                                               *
                ;* List is the New list device output. As implemented, it uses   *
                ;* an XON/XOFF or ETX/ACK protocal.                              *
                ;*                                                               *
                ;*****************************************************************

0358 3A3F00    NLIST:  LDA     ACKXON
035B 3D                DCR     A
035C FA3600            JM      LST
035F 3D                DCR     A
0360 FA6A03            JM      LSTETX
0363 3D                DCR     A
0364 FA8003            JM      LSTXON
0367 C33600            JMP     LST

036A C5        LSTETX: PUSH    B               ;Save the character
036B CD3600            CALL    LST             ;Print the character
036E C1                POP     B
036F 79                MOV     A,C             ;Check if it was a carriage return
0370 FE0D              CPI     ACR
0372 C0                RNZ
0373 0E03              MVI     C,AETX          ;Send an ETX
0375 CD3600            CALL    LST
0378 CD3C00    WETX:   CALL    HNDETX          ;Check if ACK
037B A7                ANA     A
037C C0                RNZ
037D C37803            JMP     WETX

0380 C5        LSTXON: PUSH    B               ;Save char to print
0381 CD3900            CALL    HNDXOF          ;Check XOFF
0384 FE01              CPI     1               ;Is it full ?
0386 CC8D03            CZ      WXOFF
0389 C1                POP     B               ;Recover char to print
038A C33600            JMP     LST

038D CD3900    WXOFF:  CALL    HNDXOF          ;Check XON
0390 FEFF              CPI     OFFH
0392 C28D03            JNZ     WXOFF
0395 C9                RET

                ;*****************************************************************
                ;*                                                               *
                ;* New list device status routine. Returns Offh if the printer   *
```

```
                ;* can except another character, otherwise it returns 0.        *
                ;*                                                               *
                ;*****************************************************************

0396 CD3900     NLSTST: CALL    HNDXOF          ;Check # of characters in queue
0399 FE01               CPI     1
039B 3E00               MVI     A,0
039D C8                 RZ                      ;Can not except another char
039E 2F                 CMA
039F C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Xonoff status. Checks if there are any characters in the      *
                ;* printers character queue. Returns with reg A = 1 if the       *
                ;* character queue is within 10 characters of being full, or     *
                ;* returns with reg A = 0ffh if the character queue is within    *
                ;* 10 characters from being empty, otherwise returns 0.          *
                ;* This can be used to implement the XON and XOFF protocal.      *
                ;*                                                               *
                ;*****************************************************************

03A0 CDC403     XONOFF: CALL    QUESIZ          ;Get number of characters in queue
03A3 EB                 XCHG
03A4 218C00             LXI     H,MAXCHRS-10
03A7 CDE203             CALL    HLCDE
03AA 3E01               MVI     A,1
03AC D8                 RC
03AD 210A00             LXI     H,10
03B0 CDE203             CALL    HLCDE
03B3 3E00               MVI     A,0
03B5 D8                 RC
03B6 2F                 CMA
03B7 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* ETX/ACK handshake routine.                                    *
                ;*                                                               *
                ;*****************************************************************

03B8 F3         ETXACK: DI
03B9 3A5900             LDA     ETXFLG
03BC 47                 MOV     B,A
03BD AF                 XRA     A
03BE 325900             STA     ETXFLG
03C1 FB                 EI
03C2 78                 MOV     A,B
03C3 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Quesiz returns the number of characters in the queue in HL.   *
                ;*                                                               *
                ;*****************************************************************
```

```
03C4 F3         QUESIZ: DI
03C5 2A9101             LHLD    QUETOP          ;Get pointer to top of queue
03C8 EB                 XCHG
03C9 2A9301             LHLD    QUEBOT          ;Get pointer to bottom of queue
03CC FB                 EI
03CD CDE203             CALL    HLCDE           ;Compare HL with DE
03D0 D2DB03             JNC     HLMDE           ;Subtract DE from HL
03D3 EB                 XCHG
03D4 CDDB03             CALL    HLMDE
03D7 EB                 XCHG
03D8 219600             LXI     H,MAXCHRS

                ;*****************************************************************
                ;*                                                               *
                ;* Hlmde subtracts DE from HL and returns.                       *
                ;*                                                               *
                ;*****************************************************************

03DB EB         HLMDE:  XCHG
03DC CDBA07             CALL    NEGHL
03DF EB                 XCHG
03E0 19                 DAD     D
03E1 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Hlcde compares HL with DE. On return the Z flag is set if     *
                ;* they are equal, the Carry flag is set if HL is less than DE.  *
                ;*                                                               *
                ;*****************************************************************

03E2 7C         HLCDE:  MOV     A,H
03E3 BA                 CMP     D
03E4 C0                 RNZ
03E5 7D                 MOV     A,L
03E6 BB                 CMP     E
03E7 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Lstdev just puts characters in the printer queue. characters  *
                ;* are removed from the queue by the print wheel interrupt       *
                ;* service routine.                                              *
                ;*                                                               *
                ;*****************************************************************

03E8 F3         LSTDEV: DI                              ;Disabled while manipulating
03E9 2A9301             LHLD    QUEBOT                  ;Get pointer to next slot
03EC 71                 MOV     M,C                     ;Insert the character
03ED 23                 INX     H                       ;Point to next slot
03EE 229301             SHLD    QUEBOT
03F1 119101             LXI     D,QUEUE+MAXCHRS         ;Address of first byte beyon
03F4 CDE203             CALL    HLCDE                   ;Compare HL with DE
03F7 C20004             JNZ     LSTDON                  ;No match, don't wrap around
03FA 21FB00             LXI     H,QUEUE                 ;First address in queue
03FD 229301             SHLD    QUEBOT
```

```
0400 3A4000    LSTDON: LDA     BASE                    ;Base address of Mult I/O
0403 C607              ADI     GRPSEL                  ;Group select 0
0405 0600              MVI     B,0
0407 CDAD07            CALL    OUTPUT
040A 3A4000            LDA     BASE
040D C605              ADI     PIC1                    ;8259 mask register
040F CDB407            CALL    INPUT                   ;Get current mask contents
0412 E6BF              ANI     OFFH-IMASK              ;Turn on print wheel interup
0414 47                MOV     B,A
0415 3A4000            LDA     BASE
0418 C605              ADI     PIC1
041A CDAD07            CALL    OUTPUT
041D FB                EI
041E C9                RET

                ;*****************************************************************
                ;*                                                               *
                ;* Restore routine. Restore should be executed to reset the      *
                ;* printer into a known state, and initialize all the ram        *
                ;* dynamic data locations.                                       *
                ;*                                                               *
                ;* Restore assumes that the 8259 interupt controller on the      *
                ;* Mult I/O board has already been initialized.                  *
                ;*                                                               *
                ;*****************************************************************

041F F3        RESTOR: DI                              ;No interupts

                ;*
                ;* If the Mult I/O board 8259 has not yet been initialized, then
                ;* use the following sequence.
                ;*
0420 11B501            LXI     D,TABLE
0423 21B501            LXI     H,TABLE
0426 7D                MOV     A,L
0427 E6E0              ANI     0E0H                    ;Form 32 byte boundry
0429 6F                MOV     L,A
042A E5                PUSH    H
042B 0E20              MVI     C,32
042D 1A        SLOOP:  LDAX    D
042E 77                MOV     M,A
042F 23                INX     H
0430 13                INX     D
0431 0D                DCR     C
0432 C22D04            JNZ     SLOOP
0435 E1                POP     H

0436 0600              MVI     B,0
0438 3A4000            LDA     BASE
043B C607              ADI     GRPSEL
043D CDAD07            CALL    OUTPUT
0440 7D                MOV     A,L
0441 F61E              ORI     D4+LTIM+ADDI+SNGL
0443 47                MOV     B,A
0444 3A4000            LDA     BASE
0447 C604              ADI     PIC0
```

```
0449 CDAD07             CALL    OUTPUT
044C 44                 MOV     B,H
044D 3A4000             LDA     BASE
0450 C605               ADI     PIC1
0452 CDAD07             CALL    OUTPUT
C  5 06FF               MVI     B,OFFH
04?7 3A4000             LDA     BASE
045A C605               ADI     PIC1
045C CDAD07             CALL    OUTPUT
                ;*
                ;* End of 8259 initialization
                ;*
045F 3A4000             LDA     BASE            ;Select group zero
0462 C607               ADI     GRPSEL
0464 0600               MVI     B,0
0466 CDAD07             CALL    OUTPUT
0469 3A4000             LDA     BASE            ;Get base I/O port
046C 067F               MVI     B,OFFH-RSTBIT   ;Low bit on restore, others high
046E CDAD07             CALL    OUTPUT          ;Output data in register B
0471 3A4000             LDA     BASE            ;Base I/O port
0474 06FF               MVI     B,-1            ;Output Restore bit high
0476 CDAD07             CALL    OUTPUT          ;Output data in register B
0479 2A4300             LHLD    DSPACE          ;Characters per inch
047C EB                 XCHG                    ;DE = characters per inch
047D 217800             LXI     H,HINC          ;HL = maximum increments per inch
0480 CDC207             CALL    HLDDE           ;Divide Hl by DE
0483 224800             SHLD    HMI             ;Save hmi = 120/(characters per inch
0486 2A4500             LHLD    DLINES          ;Lines per inch
0489 EB                 XCHG                    ;DE = lines per inch
048A 213000             LXI     H,VINC          ;HL = MAximum increments per inch
0?8D CDC207             CALL    HLDDE           ;Divide HL by DE
C  3 224A00             SHLD    VMI             ;Save vmi = 48/(lines per inch)
0493 210000             LXI     H,0             ;Other variables default to zero
0496 224C00             SHLD    VPOS
0499 224E00             SHLD    DLVPOS
049C 225000             SHLD    HPOS
049F 225200             SHLD    DLHPOS
04A2 225400             SHLD    LMAR
04A5 AF                 XRA     A
04A6 325600             STA     DIRFLG
04A9 325700             STA     GRHFLG
04AC 325800             STA     ESCFLG
04AF 325A00             STA     HNDFLG

04B2 21FB00             LXI     H,QUEUE         ;Zero the command queue
04B5 229101             SHLD    QUETOP
04B8 229301             SHLD    QUEBOT

04BB 0666               MVI     B,EOI6          ;Specific end of interupt 6
04BD 3A4000             LDA     BASE
04C0 C604               ADI     PIC0
04C2 CDAD07             CALL    OUTPUT
04C5 3A4000             LDA     BASE            ;Get the interupt mask bits
04C8 C605               ADI     PIC1
04CA CDB407             CALL    INPUT
04CD E6BF               ANI     OFFH-IMASK      ;Enable the daisy port interupt
```

```
04CF 47                 MOV     B,A
04D0 3A4000             LDA     BASE
04D3 C605               ADI     PIC1
04D5 CDAD07             CALL    OUTPUT          ;Output the daisy port interupt mask

04D8 FB                 EI                      ;Ok for interupts now

                ;*****************************************************************
                ;*                                                               *
                ;* Clear all tab stops.                                          *
                ;*                                                               *
                ;*****************************************************************

04D9 215B00     NOTABS: LXI     H,TABSTP                ;Beginning of tab stop array
04DC 11A000             LXI     D,NUMTABS               ;Number of tab stops
04DF 3600       NOTBLP: MVI     M,0                     ;Reset the tab
04E1 23                 INX     H                       ;Next tab stop
04E2 1B                 DCX     D                       ;Update repeat count
04E3 7B                 MOV     A,E                     ;Test for zero
04E4 B2                 ORA     D
04E5 C2DF04             JNZ     NOTBLP                  ;Continue zeroing
04E8 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Noint should never be executed. If it is then just die.       *
                ;*                                                               *
                ;*****************************************************************

04E9 C3E904     NOINT:  JMP     NOINT                   ;Die in jump self

                ;*****************************************************************
                ;*                                                               *
                ;* Pwint is the interupt service routine for the Hityp II.       *
                ;* Remember: interupts are disabled.                             *
                ;*                                                               *
                ;*****************************************************************

04EC 32D701     PWINT:  STA     AFSAVE          ;Save the acumulator
04EF 22D501             SHLD    HLSAVE          ;Save HL
04F2 17                 RAL                     ;Get the carry into register A
04F3 210000             LXI     H,0
04F6 39                 DAD     SP              ;Get the Stack pointer
04F7 31F801             LXI     SP,STACK        ;Set up new stack
04FA E5                 PUSH    H               ;Save old stack pointer
04FB 1F                 RAR                     ;Restore the carry
04FC 3AD701             LDA     AFSAVE          ;Get original contents of acumulator
04FF F5                 PUSH    PSW             ;Save acc
0500 2AD501             LHLD    HLSAVE          ;Get original contents of HL
0503 E5                 PUSH    H               ;Save HL
0504 C5                 PUSH    B               ;Save BC
0505 D5                 PUSH    D               ;Save DE
0506 3A4000             LDA     BASE            ;Select group zero
0509 C607               ADI     GRPSEL
050B 0600               MVI     B,0
050D CDAD07             CALL    OUTPUT
```

```
0510 2A9301             LHLD    QUEBOT          ;Get bottom of queue
0513 EB                 XCHG
0514 2A9101             LHLD    QUETOP          ;Get top of queue
0517 CDE203             CALL    HLCDE           ;Is there anything in the queue ?
051A CA4C05             JZ      EMPTY           ;No, queue is empty
051D 4E                 MOV     C,M             ;Get the next character
051E 23                 INX     H               ;Bump queue pointer
051F 229101             SHLD    QUETOP          ;Save the adjusted queue top
0522 119101             LXI     D,QUEUE+MAXCHRS ;Address of byte past queue
0525 CDE203             CALL    HLCDE           ;Need to wrap ?
0528 C23105             JNZ     PWDON
052B 21FB00             LXI     H,QUEUE         ;Adjust queue top
052E 229101             SHLD    QUETOP
0531 CD6805     PWDON:  CALL    DIABLO          ;Process the character
0534 0666       INTRET: MVI     B,EOI6          ;End of interupt service routine
0536 3A4000             LDA     BASE
0539 C604               ADI     PICO
053B CDAD07             CALL    OUTPUT
053E D1                 POP     D               ;Restore DE
053F C1                 POP     B               ;Restore BC
0540 E1                 POP     H               ;Get original HL
0541 F1                 POP     PSW             ;Restore PSW
0542 22D501             SHLD    HLSAVE          ;Save HL
0545 E1                 POP     H               ;Get original SP
0546 F9                 SPHL                    ;Restore original SP
0547 2AD501             LHLD    HLSAVE          ;Restore HL
054A FB                 EI                      ;Turn interupts back on
054B C9                 RET                     ;Go back

                ;*****************************************************************
                ;*                                                               *
                ;* Empty turns off the print wheel interupt mask bit if the      *
                ;* character queue is empty when an interupt occurs.             *
                ;*                                                               *
                ;*****************************************************************

054C CD7808     EMPTY:  CALL    PAPER           ;Print any remaining motion
054F CD0408             CALL    CARRG
0552 3A4000             LDA     BASE            ;Base of Mult I/O
0555 C605               ADI     PIC1            ;Get the interupt mask register
0557 CDB407             CALL    INPUT           ;Read the current mask
055A F640               ORI     IMASK           ;Turn on the bit
055C 47                 MOV     B,A             ;Data into B
055D 3A4000             LDA     BASE            ;Put the mask back
0560 C605               ADI     PIC1
0562 CDAD07             CALL    OUTPUT
0565 C33405             JMP     INTRET

                ;*****************************************************************
                ;*                                                               *
                ;* Diablo does all of the character decoding, escape sequences   *
                ;* forward, backward, etc. The list of escape sequences, and     *
                ;* special characters recognized is:                             *
                ;*                                                               *
                ;*      adel            ignored                                  *
                ;*      anul            ignored                                  *
```

```
                ;*      aack            ignored (when received)                 *
                ;*      abel            ignored                                 *
                ;*      aff             form feed                               *
                ;*      aetx            etx/ack handshake                       *
                ;*      aht             horizontal tab                          *
                ;*      alf             line feed                               *
                ;*      asp             space                                   *
                ;*      abs             backspace                               *
                ;*      acr             carriage return                         *
                ;*      aesc 0          ignored                                 *
                ;*      aesc 1          set tab stop at current print position  *
                ;*      aesc 2          clear all tab stops                     *
                ;*      aesc 3          graphics mode on                        *
                ;*      aesc 4          graphics mode off                       *
                ;*      aesc 5          forward print                           *
                ;*      aesc 6          backward print                          *
                ;*      aesc 8          clear tab stop                          *
                ;*      aesc 9          set left margin                         *
                ;*      aesc A          ignored                                 *
                ;*      aesc B          ignored                                 *
                ;*      aesc D          negative half line feed                 *
                ;*      aesc U          half line feed                          *
                ;*      aesc alf        negative line feed                      *
                ;*      aesc aht c      absolute horizontal tab                 *
                ;*      aesc avt c      absolute vertical tab                   *
                ;*      aesc ars c      set vmi                                 *
                ;*      aesc aus c      set hmi                                 *
                ;*                                                              *
                ;****************************************************************

0568 79         DIABLO: MOV     A,C             ;Get the character to print
0569 E67F               ANI     7FH             ;Strip off parity
056B C8                 RZ
056C FE7F               CPI     ADEL            ;Ignore delete
056E C8                 RZ
056F 4F                 MOV     C,A             ;Save character
0570 3A5800             LDA     ESCFLG
0573 21A205             LXI     H,LEVEL0        ;Level zero characters
0576 A7                 ANA     A
0577 79                 MOV     A,C             ;Scan for char in A
0578 CA8D05             JZ      LOOKUP          ;Look up activity for this character
057B 3A5800             LDA     ESCFLG
057E 21BD05             LXI     H,LEVEL1        ;Single character escae sequences
0581 FE1B               CPI     AESC
0583 79                 MOV     A,C             ;Scan for char in A
0584 CA8D05             JZ      LOOKUP          ;Execute single level escape sequenc
0587 21FC05             LXI     H,LEVEL2        ;Two character escape sequence
058A 3A5800             LDA     ESCFLG

                ;****************************************************************
                ;*                                                              *
                ;* Lookup scans the table pointed at by HL looking for a match  *
                ;* of the character in register A.                              *
                ;*                                                              *
                ;****************************************************************
```

```
058D 35        LOOKUP: DCR     M               ;Test if end of table
058E 34                INR     M
058F CA9C05            JZ      GOTHER          ;Execute the default function
0592 BE                CMP     M               ;Otherwise test for a match
0593 CA9C05            JZ      GOTHER
   6 23                INX     H               ;Bump over character
0597 23                INX     H               ;Bump over function address
0598 23                INX     H
0599 C38D05            JMP     LOOKUP
059C 23        GOTHER: INX     H               ;Bump over character
059D 7E                MOV     A,M             ;Get low byte of function address
059E 23                INX     H
059F 66                MOV     H,M             ;Get high byte of function address
05A0 6F                MOV     L,A             ;Form Address of function
05A1 E9                PCHL                    ;Execute it

                ;*****************************************************************
                ;*                                                               *
                ;* Each of the following tables contains entries of the form:    *
                ;*      1 byte character to match                                *
                ;*      2 bytes of address to execute                            *
                ;* terminated by a first byte of 0.                              *
                ;*                                                               *
                ;*****************************************************************

05A2 1B        LEVEL0: DB      AESC
05A3 0B06              DW      DOAESC          ;Beginning of an escape sequence
05A5 0C                DB      AFF
05A6 7F07              DW      DOAFF           ;Form feed
05A8 03                DB      AETX
05A9 1006              DW      DOAETX
   B 09                DB      AHT
05AC 5507              DW      DOAHT           ;horizontal tab
05AE 0A                DB      ALF
05AF 1606              DW      DOALF           ;Line feed
05B1 20                DB      ASP
05B2 3A06              DW      DOASP           ;Space
05B4 08                DB      ABS
05B5 5906              DW      DOABS           ;Back space
05B7 0D                DB      ACR
05B8 6206              DW      DOACR           ;Carriage return
05BA 00                DB      0
05BB 7E06              DW      DOCHAR          ;Any other character

05BD 31        LEVEL1: DB      '1'
05BE 3007              DW      SETHTAB         ;Set horizontal tab
05C0 32                DB      '2'
05C1 9406              DW      CLRALL          ;Clear all horizontal tabs
05C3 33                DB      '3'
05C4 9C06              DW      SETGRP          ;Graphics mode
05C6 34                DB      '4'
05C7 A406              DW      CLRGRP          ;Clear graphics mode
05C9 35                DB      '5'
05CA AB06              DW      CLRDIR          ;Forward printing
05CC 36                DB      '6'
05CD B206              DW      SETDIR          ;Backward printing
```

```
05CF 38                 DB      '8'
05D0 4D07               DW      CLRHTAB         ;Clear horizontal tab
05D2 39                 DB      '9'
05D3 BA06               DW      SETLMAR         ;Set left margin
05D5 30                 DB      '0'
05D6 9706               DW      FUNC1           ;No operation level 1
05D8 41                 DB      'A'
05D9 9706               DW      FUNC1
05DB 42                 DB      'B'
05DC 9706               DW      FUNC1
05DE 61                 DB      'a'
05DF 9706               DW      FUNC1
05E1 62                 DB      'b'
05E2 9706               DW      FUNC1
05E4 44                 DB      'D'
05E5 E506               DW      NEGHLF          ;Negative half line feed
05E7 55                 DB      'U'
05E8 DC06               DW      POSHLF          ;Half line feed
05EA 0A                 DB      ALF
05EB 2E06               DW      NEGLF           ;Negative line feed
05ED 09                 DB      AHT
05EE 0B06               DW      SETTWO          ;Two character escape sequence
05F0 0B                 DB      AVT
05F1 0B06               DW      SETTWO
05F3 1E                 DB      ARS
05F4 0B06               DW      SETTWO
05F6 1F                 DB      AUS
05F7 0B06               DW      SETTWO
05F9 00                 DB      0
05FA 9706               DW      FUNC1

05FC 09         LEVEL2: DB      AHT
05FD FC06               DW      ABSHTAB         ;Absolute horizontal tab
05FF 0B                 DB      AVT
0600 1807               DW      ABSVTAB         ;Absolute vertical tab
0602 1E                 DB      ARS
0603 C806               DW      SETVMI
0605 1F                 DB      AUS
0606 D206               DW      SETHMI
0608 00                 DB      0
0609 9706               DW      FUNC2

                ;*****************************************************************
                ;*                                                               *
                ;* The following routines execute escape sequences, etc.         *
                ;*                                                               *
                ;*****************************************************************

                SETTWO:
060B 79         DOAESC: MOV     A,C             ;Get the escape character
060C 325800             STA     ESCFLG
060F C9         FUNC0:  RET

0610 3EFF       DOAETX: MVI     A,0FFH          ;Set the handshake flag
0612 325900             STA     ETXFLG
0615 C9                 RET
```

```
0616 CD2206     DOALF:  CALL    LFVMI           ;Get line feed vmi
0619 EB         ADJVP:  XCHG
061A 2A4E00             LHLD    DLVPOS          ;Get vertical motion displacement
061D 19                 DAD     D
061E 224E00             SHLD    DLVPOS
0621 C9                 RET

0622 3A5700     LFVMI:  LDA     GRHFLG
0625 A7                 ANA     A
0626 210100             LXI     H,1             ;Only 1/48 if in graphics mode
0629 C0                 RNZ
062A 2A4A00             LHLD    VMI             ;Get vertical motion index
062D C9                 RET

062E CD2206     NEGLF:  CALL    LFVMI           ;Get line feed vmi
0631 CDBA07             CALL    NEGHL
0634 CD1906             CALL    ADJVP
0637 C39706             JMP     FUNC1

063A CD4D06     DOASP:  CALL    SPHMI           ;Get space horizontal motion
063D 3A5600     SPDIR:  LDA     DIRFLG          ;Forward or backwards ?
0640 A7                 ANA     A
0641 C4BA07             CNZ     NEGHL           ;Negate HL
0644 EB         ADJHP:  XCHG                    ;Adjust Horizontal position
0645 2A5200             LHLD    DLHPOS          ;Get current adjustment
0648 19                 DAD     D               ;Update it
0649 225200             SHLD    DLHPOS          ;And save
064C C9                 RET

064D 3A5700     SPHMI:  LDA     GRHFLG          ;In graphics mode ?
0650 A7                 ANA     A
0651 210200             LXI     H,2             ;Only 1/60 if in graphics mode
0654 C0                 RNZ
0655 2A4800             LHLD    HMI
0658 C9                 RET

0659 CD4D06     DOABS:  CALL    SPHMI           ;Space increment
065C CDBA07             CALL    NEGHL           ;Negative to start with
065F C33D06             JMP     SPDIR           ;Adjust backwards

0662 AF         DOACR:  XRA     A
0663 325600             STA     DIRFLG          ;Forward printing
0666 325700             STA     GRHFLG          ;No graphics mode
0669 2A5000             LHLD    HPOS            ;Get current offset
066C EB                 XCHG
066D 2A5400             LHLD    LMAR            ;Get left margin
0670 CDDB03             CALL    HLMDE
0673 225200             SHLD    DLHPOS          ;Don't move yet though
0676 3A4700             LDA     AUTOLF          ;In Auto line feed mode ?
0679 A7                 ANA     A
067A C21606             JNZ     DOALF           ;Do line feed also
067D C9                 RET

067E 69         DOCHAR: MOV     L,C
067F 2600               MVI     H,0
```

```
0681 CDBE08            CALL    WHEEL           ;Print the character in register C
0684 3A5700            LDA     GRHFLG
0687 A7                ANA     A
0688 210000            LXI     H,0             ;Don't move if in graphics mode
068B C23D06            JNZ     SPDIR
068E 2A4800            LHLD    HMI
0691 C33D06            JMP     SPDIR

0694 CDD904    CLRALL: CALL    NOTABS          ;Clear all horizontal tabs
               FUNC2:
0697 AF        FUNC1:  XRA     A               ;Clear escape sequence flag
0698 325800            STA     ESCFLG
069B C9                RET

069C 3E01      SETGRP: MVI     A,1             ;Set graphics mode on
069E 325700            STA     GRHFLG
06A1 C39706            JMP     FUNC1

06A4 AF        CLRGRP: XRA     A               ;Turn graphics mode off
06A5 325700            STA     GRHFLG
06A8 C39706            JMP     FUNC1

06AB AF        CLRDIR: XRA     A               ;Forward print mode
06AC 325600            STA     DIRFLG
06AF C39706            JMP     FUNC1

06B2 3E07      SETDIR: MVI     A,A             ;Set backward printing mode
06B4 325600            STA     DIRFLG
06B7 C39706            JMP     FUNC1

06BA 2A5000    SETLMAR: LHLD   HPOS            ;Get current position
06BD EB                XCHG
06BE 2A5200            LHLD    DLHPOS          ;Get offset
06C1 19                DAD     D
06C2 225400            SHLD    LMAR
06C5 C39706            JMP     FUNC1

06C8 69        SETVMI: MOV     L,C             ;Set the motion index
06C9 2600              MVI     H,0
06CB 2B                DCX     H
06CC 224A00            SHLD    VMI
06CF C39706            JMP     FUNC2

06D2 69        SETHMI: MOV     L,C
06D3 2600              MVI     H,0
06D5 2B                DCX     H
06D6 224800            SHLD    HMI
06D9 C39706            JMP     FUNC2

06DC CDF106    POSHLF: CALL    HLFVMI          ;Half line feed vmi
06DF CD1906            CALL    ADJVP
06E2 C39706            JMP     FUNC1

06E5 CDF106    NEGHLF: CALL    HLFVMI          ;Negative half line feed
06E8 CDBA07            CALL    NEGHL
06EB CD1906            CALL    ADJVP
```

```
06EE C39706             JMP     FUNC1

06F1 2A4A00     HLFVMI: LHLD    VMI             ;Get vmi for full line feed
06F4 7C         DIVID2: MOV     A,H             ;High byte
06F5 B7                 ORA     A               ;Clear the carry
06F6 1F                 RAR
06F7 67                 MOV     H,A
06F8 7D                 MOV     A,L
06F9 1F                 RAR
06FA 6F                 MOV     L,A
06FB C9                 RET

06FC 59         ABSHTAB: MOV    E,C             ;Absolute horizontal tab
06FD 1600               MVI     D,0
06FF 1B                 DCX     D               ;Form 16 bit tab column
0700 CD0607             CALL    NEWDLH
0703 C39706             JMP     FUNC2

0706 2A4800     NEWDLH: LHLD    HMI
0709 CDEA07             CALL    HLTDE           ;Multiply by hmi
070C EB                 XCHG
070D 2A5000             LHLD    HPOS            ;And subtract current horizontal pos
0710 EB                 XCHG
0711 CDDB03             CALL    HLMDE
0714 225200             SHLD    DLHPOS
0717 C9                 RET

0718 59         ABSVTAB: MOV    E,C             ;Absolute vertical tab
0719 1600               MVI     D,0
071B 1B                 DCX     D
071C 2A4A00             LHLD    VMI
071F CDEA07             CALL    HLTDE           ;Multiply by vmi
0722 EB                 XCHG
0723 2A4C00             LHLD    VPOS            ;And subtract the current vertical p
0726 EB                 XCHG
0727 CDDB03             CALL    HLMDE
072A 224E00             SHLD    DLVPOS
072D C39706             JMP     FUNC2

0730 CD3807     SETHTAB: CALL   TABCOL          ;Set horizontal tab
0733 3601               MVI     M,1
0735 C39706             JMP     FUNC1

0738 2A5000     TABCOL: LHLD    HPOS            ;Compute address of current characte
073B EB                 XCHG
073C 2A5200             LHLD    DLHPOS
073F 19                 DAD     D               ;Get logical position
0740 EB                 XCHG
0741 2A4800             LHLD    HMI             ;And divide by hmi to get character
0744 EB                 XCHG
0745 CDC207             CALL    HLDDE
0748 115B00             LXI     D,TABSTP
074B 19                 DAD     D               ;Index into the tab stop array
074C C9                 RET

074D CD3807     CLRHTAB: CALL   TABCOL          ;Clear horizontal tab
```

```
0750 3600              MVI     M,0
0752 C39706            JMP     FUNC1

0755 CD3807    DOAHT:  CALL    TABCOL          ;Get current tab column
0758 11FB00            LXI     D,TABSTP+NUMTABS
075B 23        TABLOP: INX     H               ;Start with next position
075C CDE203            CALL    HLCDE
075F D27107            JNC     TOFAR           ;Past last tab
0762 7E                MOV     A,M             ;Get value of current column
0763 A7                ANA     A               ;Test if it is set
0764 CA5B07            JZ      TABLOP
0767 115B00            LXI     D,TABSTP        ;Subtract off array address
076A CDDB03            CALL    HLMDE
076D EB                XCHG
076E C30607            JMP     NEWDLH
0771 2A5000    TOFAR:  LHLD    HPOS
0774 EB                XCHG
0775 213006            LXI     H,MAXRGT
0778 CDDB03            CALL    HLMDE
077B 225200            SHLD    DLHPOS
077E C9                RET

077F 2A4100    DOAFF:  LHLD    DFRMLN          ;Multiply forms length by 48
0782 113000            LXI     D,48
0785 CDEA07            CALL    HLTDE
0788 110A00            LXI     D,10
078B CDC207            CALL    HLDDE           ;And divide it by 10
078E E5                PUSH    H               ;Save this result
078F 2A4C00            LHLD    VPOS            ;Get logical vertical position
0792 EB                XCHG
0793 2A4E00            LHLD    DLVPOS
0796 19                DAD     D
0797 D1                POP     D
0798 D5                PUSH    D               ;Get copy of forms length
0799 CDC207            CALL    HLDDE           ;HL mod DE
079C EB                XCHG
079D D1                POP     D
079E EB                XCHG
079F CDDB03            CALL    HLMDE
07A2 EB                XCHG
07A3 2A4E00            LHLD    DLVPOS
07A6 19                DAD     D
07A7 224E00            SHLD    DLVPOS
07AA C37808            JMP     PAPER

                ;*****************************************************************
                ;*                                                               *
                ;* Output the data in register B to the port in register A.      *
                ;*                                                               *
                ;*****************************************************************

07AD 32B207    OUTPUT: STA     OUTNUM          ;Put port number in the instruction
07B0 78                MOV     A,B             ;Data to register A.
07B1 D300              OUT     0               ;Self modified to port number
07B2 =         OUTNUM  EQU     $-1
07B3 C9                RET
```

```
                ;*****************************************************************
                ;*                                                               *
                ;* Input from the port in register A.                            *
                ;*                                                               *
                ;*****************************************************************

07B4 32B807     INPUT:  STA     INNUM           ;Put port number in the instruction
07B7 DB00               IN      0               ;Self modified port number
07B8 =          INNUM   EQU     $-1
07B9 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Neghl forms the twos complement of HL.                        *
                ;*                                                               *
                ;*****************************************************************

07BA 7C         NEGHL:  MOV     A,H
07BB 2F                 CMA
07BC 67                 MOV     H,A
07BD 7D                 MOV     A,L
07BE 2F                 CMA
07BF 6F                 MOV     L,A
07C0 23                 INX     H
07C1 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* Divide the number in HL by the number in DE. Return the       *
                ;* quotient in HL and the remainder in DE.                       *
                ;*                                                               *
                ;*****************************************************************

07C2 7A         HLDDE:  MOV     A,D             ;Start by negating DE and
07C3 2F                 CMA                     ;       moving the left operand to B
07C4 47                 MOV     B,A
07C5 7B                 MOV     A,E
07C6 2F                 CMA
07C7 4F                 MOV     C,A
07C8 03                 INX     B
07C9 3E10               MVI     A,16            ;Repeat count in reg A
07CB 110000             LXI     D,0             ;Initial remainder is zero
07CE 3D         DIV3:   DCR     A               ;Test if done
07CF F8                 RM                      ;All done ?
07D0 29                 DAD     H               ;Shift right operand to the left
07D1 EB                 XCHG
07D2 F5                 PUSH    PSW             ;Save carry
07D3 29                 DAD     H               ;Shift left operand to the left
07D4 F1                 POP     PSW
07D5 D2D907             JNC     DIV1            ;Does it fit ?
07D8 23                 INX     H
07D9 E5         DIV1:   PUSH    H
07DA 09                 DAD     B
07DB D2E507             JNC     DIV2
07DE EB                 XCHG
```

```
07DF 23                 INX     H
07E0 E3                 XTHL
07E1 E1                 POP     H
07E2 C3CE07             JMP     DIV3
07E5 E1         DIV2:   POP     H
07E6 EB                 XCHG
07E7 C3CE07             JMP     DIV3

                ;*****************************************************************
                ;*                                                               *
                ;* Multiply the contents of HL by the contents of DE.            *
                ;*                                                               *
                ;*****************************************************************

07EA 4D         HLTDE:  MOV     C,L
07EB 44                 MOV     B,H
07EC 210000             LXI     H,0
07EF 78         MULT:   MOV     A,B
07F0 B1                 ORA     C
07F1 C8                 RZ
07F2 78                 MOV     A,B
07F3 B7                 ORA     A
07F4 1F                 RAR
07F5 47                 MOV     B,A
07F6 79                 MOV     A,C
07F7 1F                 RAR
07F8 4F                 MOV     C,A
07F9 DC0208             CC      DADDE
07FC EB                 XCHG
07FD 29                 DAD     H
07FE EB                 XCHG
07FF C3EF07             JMP     MULT
0802 19         DADDE:  DAD     D
0803 C9                 RET

                ;*****************************************************************
                ;*                                                               *
                ;* The routines below actually interface to the printer,         *
                ;* causing paper feed, carriage, and print wheel motion.         *
                ;*                                                               *
                ;*****************************************************************

0804 2A5200     CARRG:  LHLD    DLHPOS          ;Check for any accumulated motion
0807 7C                 MOV     A,H
0808 B5                 ORA     L
0809 C8                 RZ
080A 2A5000             LHLD    HPOS            ;Check for to much motion
080D EB                 XCHG
080E 2A5200             LHLD    DLHPOS
0811 19                 DAD     D
0812 7C                 MOV     A,H
0813 A7                 ANA     A
0814 F22008             JP      LFTOK
0817 2A5000             LHLD    HPOS
081A CDBA07             CALL    NEGHL
081D 225200             SHLD    DLHPOS
```

```
0820 2A5000     LFTOK:  LHLD    HPOS
0823 EB                 XCHG
0824 2A5200             LHLD    DLHPOS
0827 19                 DAD     D
0828 113006             LXI     D,MAXRGT
082B CDE203             CALL    HLCDE
082E DA3E08             JC      RGTOK
0831 2A5000             LHLD    HPOS            ;Otherwise move only to maxright
0834 EB                 XCHG
0835 213006             LXI     H,MAXRGT
0838 CDDB03             CALL    HLMDE
083B 225200             SHLD    DLHPOS
083E 2A5000     RGTOK:  LHLD    HPOS            ;Update the horizontal position
0841 EB                 XCHG
0842 2A5200             LHLD    DLHPOS
0845 19                 DAD     D
0846 225000             SHLD    HPOS
0849 2A5200             LHLD    DLHPOS          ;check if required motion is to the
084C 7C                 MOV     A,H
084D A7                 ANA     A
084E 0E00               MVI     C,0
0850 F25808             JP      POSH
0853 CDBA07             CALL    NEGHL
0856 0E04               MVI     C,DATA11
0858 EB         POSH:   XCHG
0859 210000             LXI     H,0
085C 225200             SHLD    DLHPOS          ;Reset the horizontal increment
085F EB                 XCHG
0860 7D                 MOV     A,L
0861 E601               ANI     1
0863 CA6A08             JZ      NOHHLF          ;No half spaces
0866 79                 MOV     A,C
0867 F608               ORI     DATA12
0869 4F                 MOV     C,A
086A CDF406     NOHHLF: CALL    DIVID2
086D 7C                 MOV     A,H
086E E603               ANI     DATA910
0870 B1                 ORA     C
0871 67                 MOV     H,A
0872 112010             LXI     D,CRSTRD
0875 C3C908             JMP     CMND

0878 2A4E00     PAPER:  LHLD    DLVPOS          ;Check for any paper motion
087B 7C                 MOV     A,H
087C B5                 ORA     L
087D C8                 RZ                      ;No motion
087E 7C                 MOV     A,H
087F A7                 ANA     A
0880 0E00               MVI     C,0
0882 F28A08             JP      POSV
0885 CDBA07             CALL    NEGHL
0888 0E04               MVI     C,DATA11
088A 7C         POSV:   MOV     A,H
088B E603               ANI     DATA910
088D B1                 ORA     C
088E 67                 MOV     H,A
```

```
088F E5              PUSH    H               ;Save paper motion
0890 2A4C00          LHLD    VPOS
0893 EB              XCHG
0894 2A4E00          LHLD    DLVPOS          ;Get logical position
0897 19              DAD     D
0898 E5              PUSH    H               ;Save for now
0899 2A4100          LHLD    DFRMLN          ;Get default form length
089C 113000          LXI     D,48
089F CDEA07          CALL    HLTDE           ;Multiply by 48
08A2 110A00          LXI     D,10
08A5 CDC207          CALL    HLDDE           ;Divide by 10
08A8 D1              POP     D
08A9 EB              XCHG
08AA CDC207          CALL    HLDDE           ;Compute HL mod DE
08AD EB              XCHG
08AE 224C00          SHLD    VPOS            ;Save new vertical position
08B1 210000          LXI     H,0
08B4 224E00          SHLD    DLVPOS          ;Reset vertical motion
08B7 E1              POP     H
08B8 111008          LXI     D,PFSTRD        ;Paper feed strobe
08BB C3C908          JMP     CMND

08BE E5        WHEEL:PUSH    H
08BF CD0408          CALL    CARRG           ;Position the carriage first
08C2 CD7808          CALL    PAPER
08C5 E1              POP     H
08C6 114020          LXI     D,PWSTRD

08C9 3A4000    CMND: LDA     BASE
08CC CDB407          CALL    INPUT
08CF A2              ANA     D
08D0 CAC908          JZ      CMND
08D3 7D              MOV     A,L
08D4 2F              CMA
08D5 6F              MOV     L,A
08D6 7C              MOV     A,H
08D7 E60F            ANI     DATA910+DATA11+DATA12
08D9 2F              CMA
08DA 67              MOV     H,A
08DB 3A4000          LDA     BASE
08DE C601            ADI     DAISY1
08E0 45              MOV     B,L
08E1 CDAD07          CALL    OUTPUT
08E4 3A4000          LDA     BASE
08E7 44              MOV     B,H
08E8 CDAD07          CALL    OUTPUT
08EB 7C              MOV     A,H
08EC AB              XRA     E
08ED 47              MOV     B,A
08EE 3A4000          LDA     BASE
08F1 CDAD07          CALL    OUTPUT
08F4 44              MOV     B,H
08F5 3A4000          LDA     BASE
08F8 C3AD07          JMP     OUTPUT

08FB                 END
```

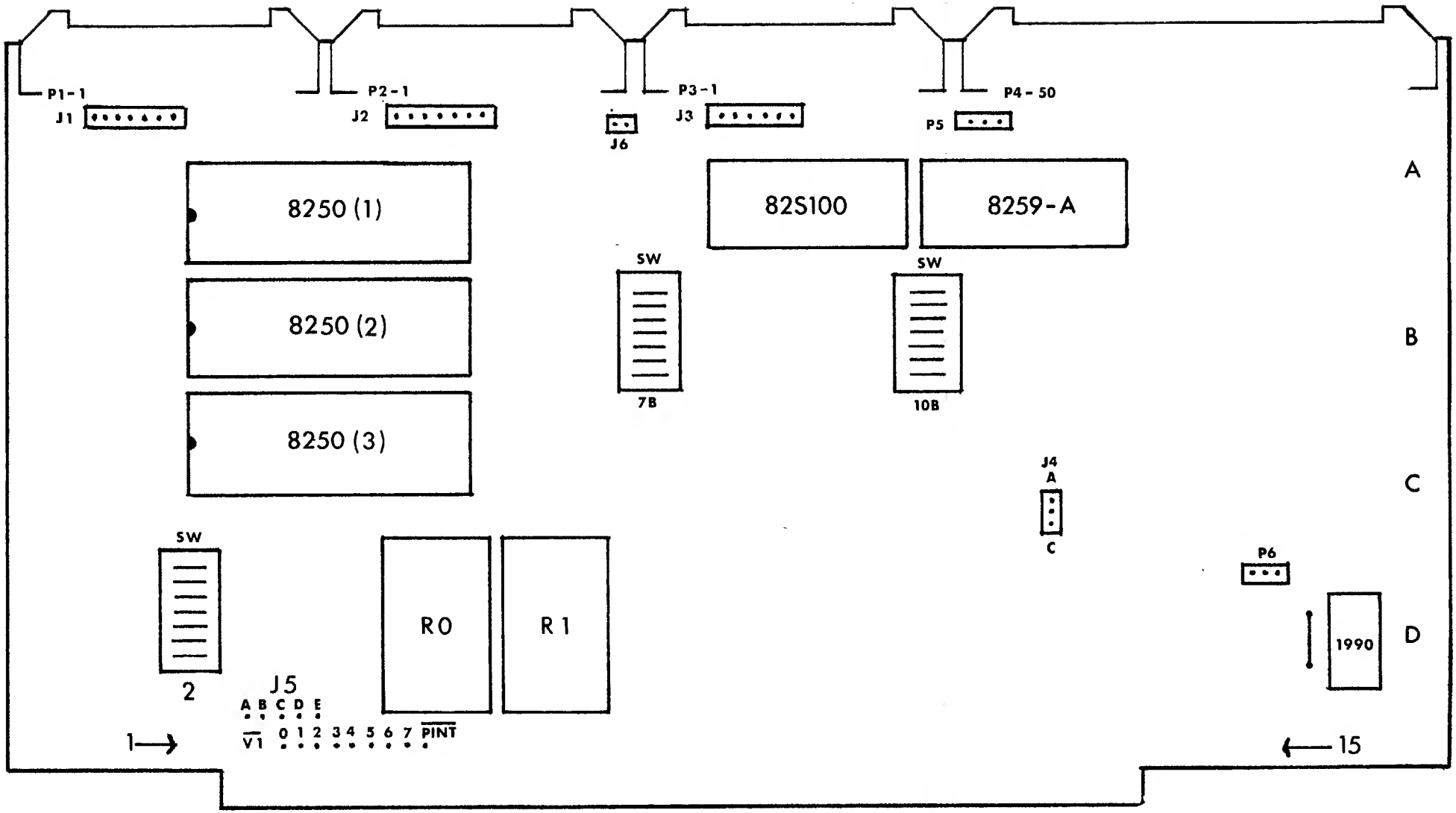
PARTS LIST, MULT/IO rev. 4

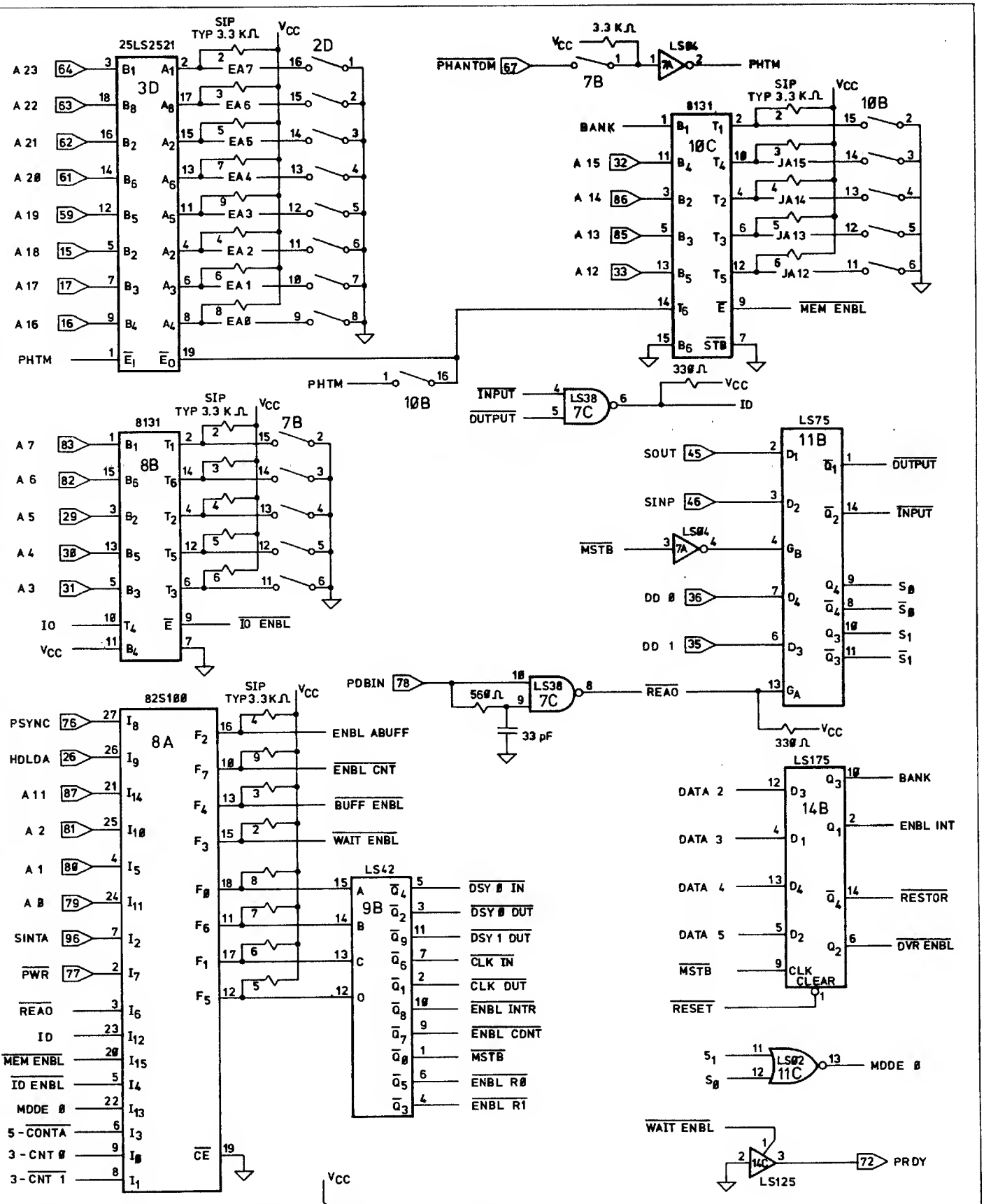
DESCRIPTION	ITEM CODE	QUANTITY
Diode 1N3600 [1N914]	Ø28-1N36ØØ	1
Transistor 2N39Ø6	Ø28-2N39Ø6	2
Transistor 2N2222	Ø28-2N2222	1
Regulator +5 volts	Ø28-78Ø5	2
Regulator +12 volt	Ø28-78L12	2
Regulator -12 volt	Ø28-79L12	2
Resistor 3.3 ohm 1/4w 5%	Ø3Ø-CØ2Ø5-Ø33	1
Resistor 1K ohm 1/4w 5%	Ø3Ø-CØ2Ø5-1Ø2	2
Resistor 1ØK ohm 1/4w 5%	Ø3Ø-CØ2Ø5-1Ø3	3
Resistor 1ØØK ohm 1/4w 5%	Ø3Ø-CØ2Ø5-1Ø5	1Ø
Resistor 1.5K ohm 1/4w 5%	Ø3Ø-CØ2Ø5-152	2
Resistor 33Ø ohm 1/4w 5%	Ø3Ø-CØ2Ø5-331	2
Resistor 3.3K ohm 1/4w 5%	Ø3Ø-CØ2Ø5-332	4
Resistor 39Ø ohm 1/4w 5%	Ø3Ø-CØ2Ø5-391	1
Resistor 4.7K ohm 1/4w 5%	Ø3Ø-CØ2Ø5-472	1
Resistor 56Ø ohm 1/4w 5%	Ø3Ø-CØ2Ø5-561	1
Sip 18Ø 1/8w 5% 8 pin	Ø3Ø-SØ1Ø5-181-Ø8	2
Sip 3.3K 1/8w 5% 8 pin	Ø3Ø-SØ1Ø5-332-Ø8	2
Sip 3.3K 1/8w 5% 1Ø pin	Ø3Ø-SØ1Ø5-332-1Ø	1
Capacitor .1 uf mono cap	Ø33-MØØØ1C	14
Capacitor 2Øpf silv mica	Ø33-SMØ2Ø	2
Capacitor 33pf silv mica	Ø33-SMØ33	1
Capacitor 56pf silv mica	Ø33-SMØ56	1
Capacitor 1ØØpf silv mica	Ø33-SM1ØØ	1
Capacitor 1 uf dip tant	Ø33-TDØ1Ø-35	1Ø
Crystal 32.768 kilo hz	Ø37-KZ32.768	1
Crystal 18.432 mega hz	Ø37-MZ18.432	1
Inductor 2.2 uh axial	Ø39-IND2.2	1
8 Position dip switch	Ø41-DSØ8	3
Slide-on connectors	Ø41-SLDJMP	2Ø
PCB Header sin str nhd 2	Ø43-Ø2SSF	1
PCB Header sin str nhd 3	Ø43-Ø3SSF	2
PCB Header sin str nhd 4	Ø43-Ø4SSF	1
PCB Header din str nhd 14	Ø43-14DSF	3
PCB Header din rt> hd 26	Ø43-26DRH	3
PCB Header din rt> hd 5Ø	Ø43-5ØDRH	1
Screw 632 x 5/ 16 pan phil	Ø96-Ø6X516PP	2
Hex nut 632	Ø98-Ø632HN	2
Heatsink low prof 3 fin	Ø94-LØ321	2
IC Socket 14 pin low prof	Ø39-SOCLP-14	9
IC Socket 16 pin low prof	Ø39-SOCLP-16	1Ø
IC Socket 2Ø pin low prof	Ø39-SOCLP-2Ø	9
IC Socket 24 pin low prof	Ø39-SOCLP-24	2
IC Socket 28 pin low prof	Ø39-SOCLP-28	2
IC Socket 4Ø pin low prof	Ø39-SOCLP-4Ø	3
I.C. 1458	Ø26-IC1458	4
I.C. 1489 [75189]	Ø26-IC1489	3
I.C. 199Ø	Ø26-IC199Ø	1
I.C. 25LS2521	Ø26-ICLS2521	1
I.C. 74LSO2	Ø26-IC74LSO2	1
I.C. 74LSO4	Ø26-IC74LSO4	2
I.C. 74LS125	Ø26-IC74LS125	1
I.C. 74LS174	Ø26-IC74LS174	1

PARTS LIST, MULT/IO rev. 4

DESCRIPTION	ITEM CODE	QUANTITY
I.C. 74LS175	Ø26-IC74LS175	1
I.C. 74LS244	Ø26-IC74LS244	4
I.C. 74LS374	Ø26-IC74LS374	2
I.C. 74LS38	Ø26-IC74LS38	1
I.C. 74LS39Ø	Ø26-IC74LS39Ø	2
I.C. 74LS42	Ø26-IC74LS42	1
I.C. 74LS75	Ø26-IC74LS75	1
I.C. 8131	Ø26-IC8131	2
I.C. 81LS95	Ø26-IC81LS95	1
I.C. 81LS96	Ø26-IC81LS96	1
I.C. 825Ø	Ø26-IC825Ø	3
I.C. 8259	Ø26-IC8259	1
I.C. 82S1ØØ FPLA REV. 3.1	Ø26-IC82S1ØØ	1

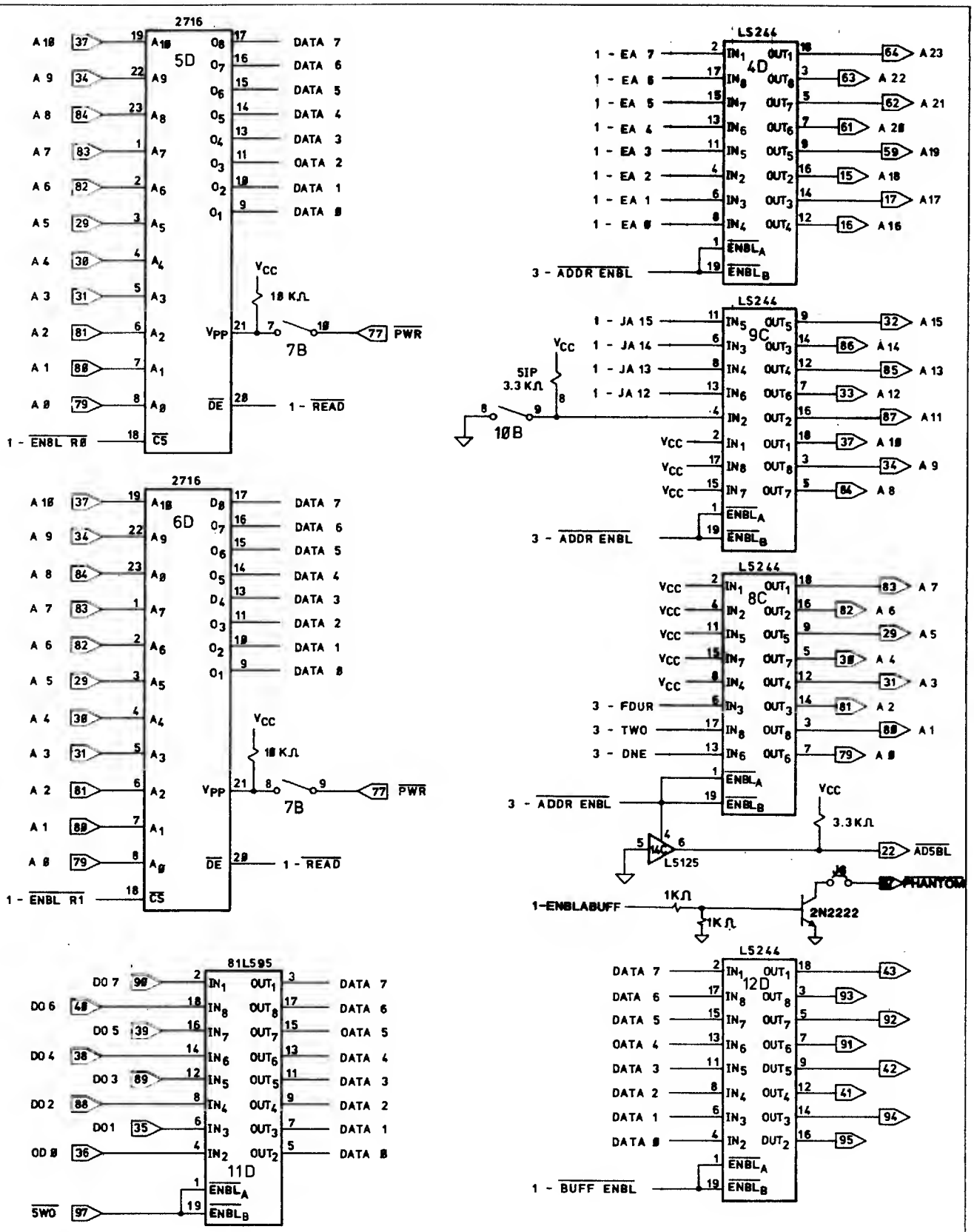
COMPONENT LAYOUT/SCHEMATICS

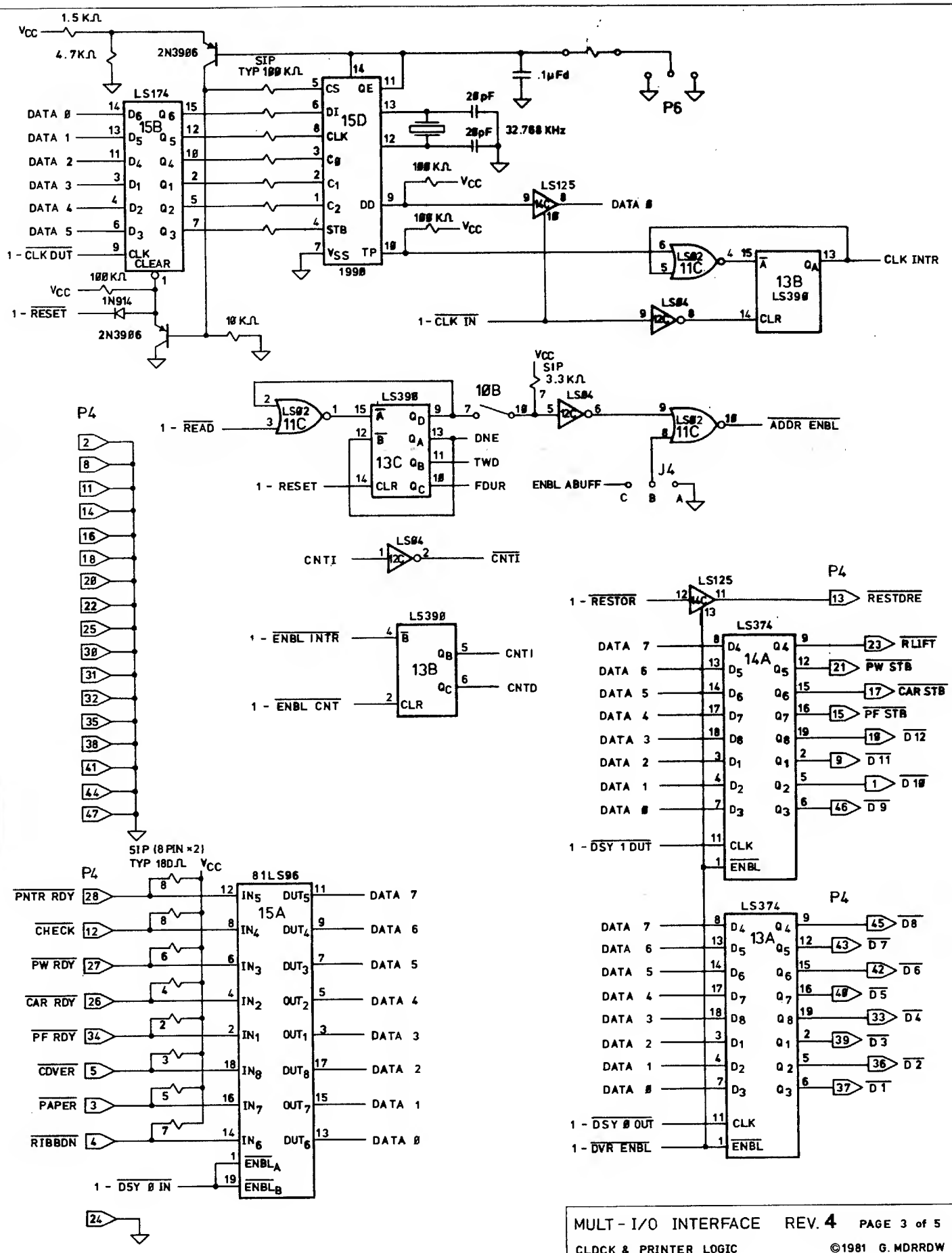

P1-1
J1
P2-1
J2
P3-1
J3
J6
P4-50
P5
8250 (1)
8250 (2)
8250 (3)
82S100
8259-A
SW
7B
SW
10B
J4
A
C
SW
2
R0
R1
J5
A B C D E
V1 0 1 2 3 4 5 6 7 PINT
1
15
P6
1990
A
B
C
D



MULT-I/O INTERFACE REV. 4 PAGE 1 of 5

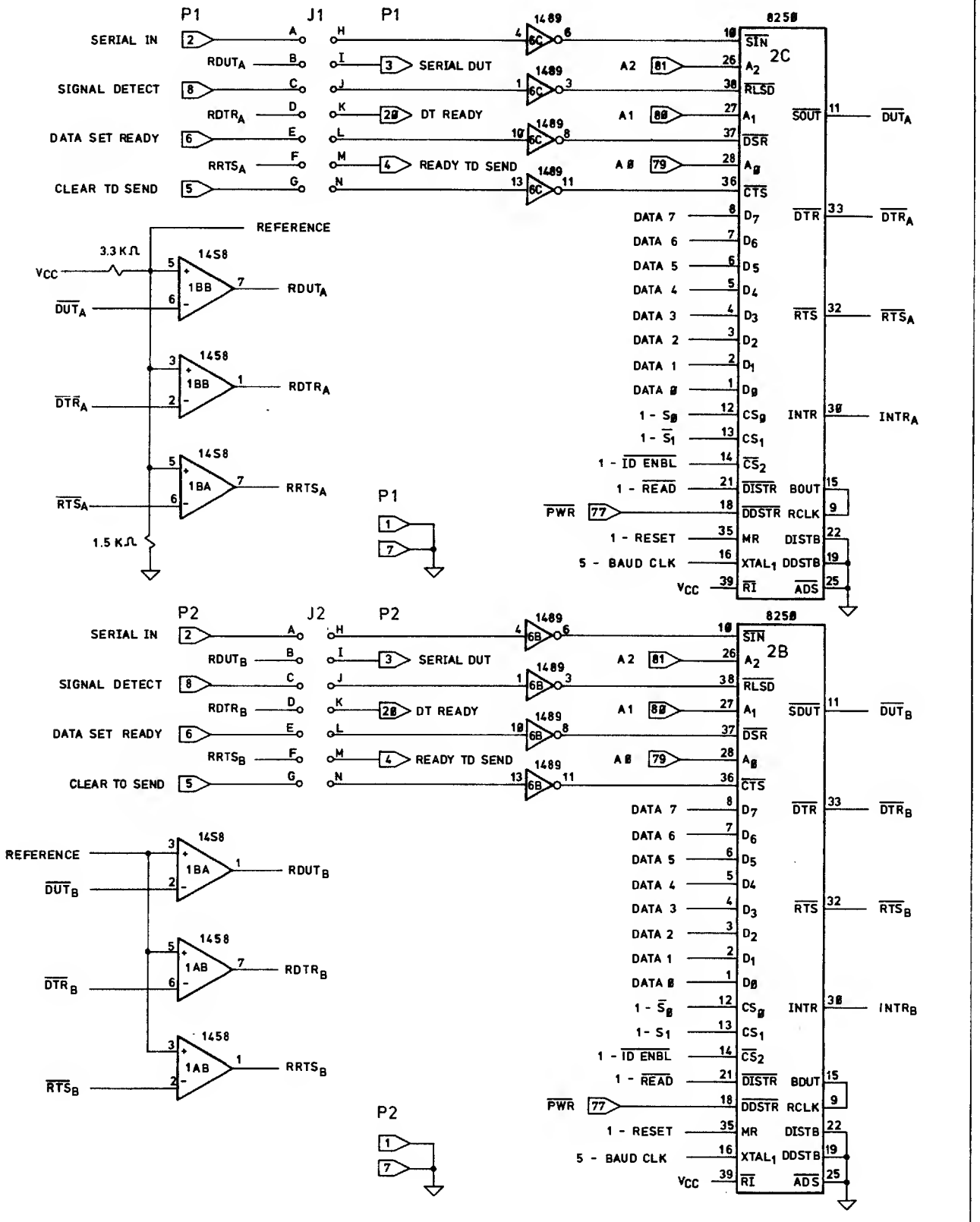
BOARD SELECT & DEVICE SELECT LOGIC ©1981 G. MORROW

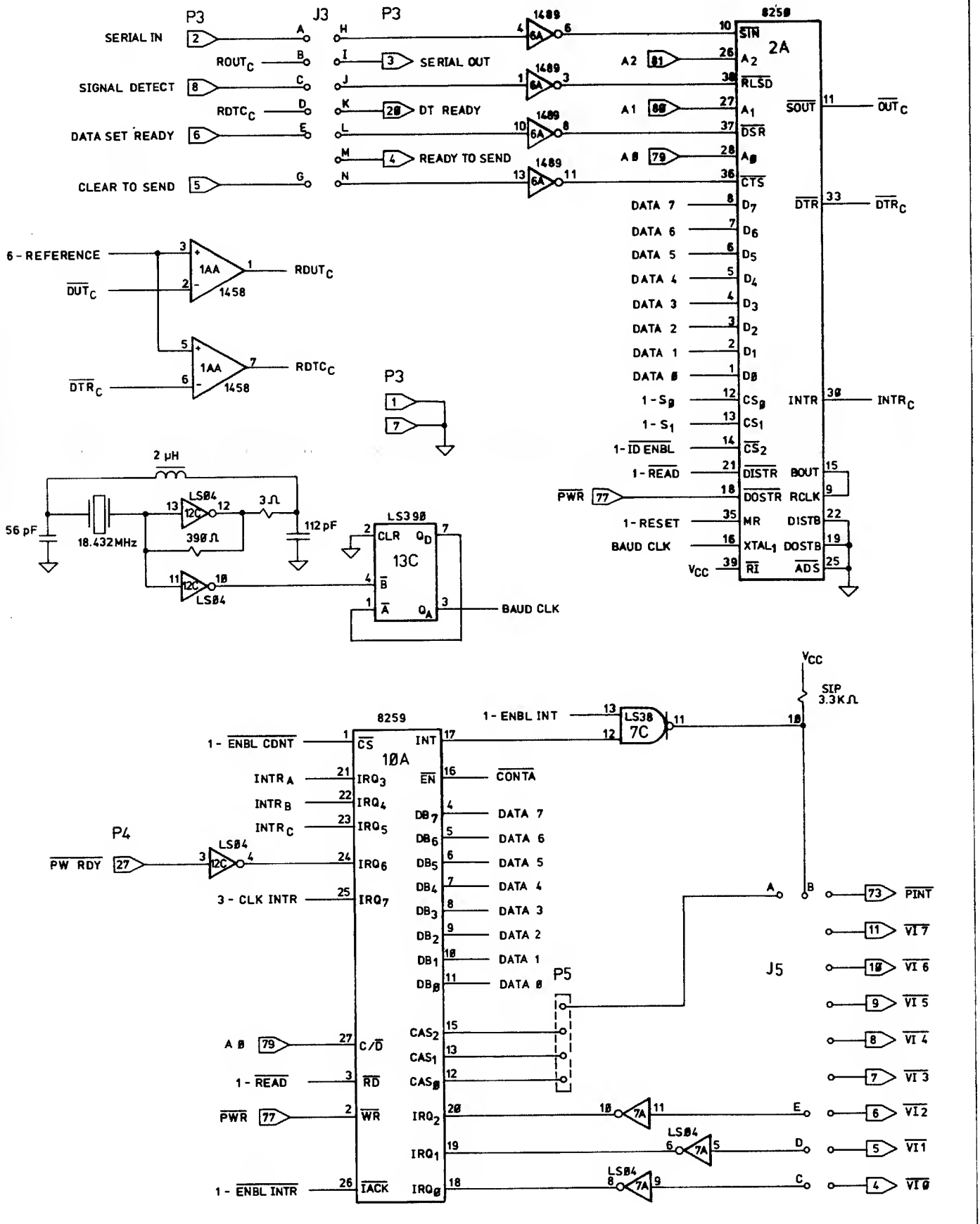




MULT - I/O INTERFACE REV. 4 PAGE 3 of 5

CLOCK & PRINTER LOGIC





MULT-I/O INTERFACE REV.4 PAGE 5 of 5

SERIAL PORT C & CONTROLLER LOGIC ©1981 G MORROW

INDEX

INDEX

INDEX